# MANAGEMENT OF PERVASIVE DISPLAYS

**Venkata Praneeth Tatiraju**

**School of Electrical Engineering**

Thesis submitted for examination for the degree of Master of Science in Technology.

Espoo 23.11.2015

**Thesis supervisor:**

Prof. Mario Di Francesco

**Thesis advisor:**

M.Sc. Mohit Sethi

Aalto University
School of Electrical
Engineering

AALTO UNIVERSITY
SCHOOL OF ELECTRICAL ENGINEERING

ABSTRACT OF THE
MASTER'S THESIS

Author: Venkata Praneeth Tatiraju

Title: MANAGEMENT OF PERVASIVE DISPLAYS

Date: 23.11.2015 Language: English Number of pages: 9+67

Department of Communications and Networking

Professorship: Data Communication Software

Supervisor: Prof. Mario Di Francesco

Advisor: M.Sc. Mohit Sethi

Traditional signage is being replaced by digital displays that are directly connected to the Internet and show content from the cloud. These displays increasingly rely on a standard web-browser and HTML5 technologies for rendering rich media content. As the number of these displays increase, it is critical to provide user-friendly and efficient solutions for managing them remotely from the cloud. The remote management of such displays traditionally relies on proprietary native software solutions that employ remote desktop access technologies such as Virtual Network Computing (VNC) and Remote Desktop Protocol (RDP). However, these solutions are not only resource-intensive in terms of the consumed bandwidth, but also cumbersome to use on mobile devices such as smartphones and tablets.

In this thesis, we design a new remote-management solution that relies on available web technologies including HTML5, WebRTC and WebSocket. In particular, we use the WebSocket protocol and a Publish/Subscribe communication pattern for our proposed solution. To demonstrate the feasibility of this remote-management solution, we implement a proof-of-concept HTML5-based application for a representative digital signage scenario. Three different versions are implemented and realized on top of state-of-the-art JavaScript libraries, namely *mutation-summary*, *sharejs*, and *socket.io*.

The performance of these solutions is evaluated in terms of payload, round trip time, throughput, and application response time. The obtained results show that mutation summary has low latency and is best suited for non-interactive content. ShareJS and Socketio are more suitable for real-time collaborative applications. Lastly, we also analyze the libraries from a programmer's perspective and present important implementation related considerations.

Keywords: WebSocket, Socket.io, ShareJS, Mutation Summary, HTML5, Cloud server, Cloning, Virtual Network Computing.

# Preface

Foremost, I am grateful to Professor Mario Di Francesco for supervising my thesis and providing me an opportunity to work with him. His excellent guidance, motivational support and feedback helped me to complete this thesis successfully. I also thank my instructor Mohit Sethi, for his patience, guidance and feedback at all stages of my thesis.
I am thankful to my friends and family for extending their support during the difficult times.
Thank You!

Otaniemi, 23.11.2015

Venkata Praneeth Tatiraju

# Abbreviations

| | |
|---|---|
| HTTP | Hypertext Transfer Protocol |
| HTTPS | Hyper Text Transfer Protocol Secure |
| WWW | World Wide Web |
| MIME | Multipurpose Internet Mail Extensions |
| HTML | Hypertext Markup Language |
| VNC | Virtual Network Computing |
| AJAX | Asynchronous JavaScript and XML |
| RFB | Remote Frame Buffer Protocol |
| TCP | Transmission Control Protocol |
| IP | Internet Protocol |
| UDP | User Datagram Protocol |
| NTP | Network Time Protocol |
| RPC | Remote Procedure Call |
| WAMP | Websocket Application Messaging Protocol |
| Pub/Sub | Publish/Subscribe |
| UA | User Agent |
| URI | Uniform Resource Identifier |
| URL | Uniform Resource Locator |
| RTT | Roundtrip Transfer Time |
| OT | Operational Transformation |
| JSON | JavaScript Object Notation |
| SOAP | Simple Object Access Protocol |
| RSS | Really Simple Syndication |
| REST | Representational State Transfer |
| XMPP | Extensible Messaging and Presence Protocol |
| XML | Extensible Markup Language |
| Wi-Fi | Wireless Fidelity |
| TLS | Transport Layer Security |
| SSH | Secure Shell |
| SSL | Secure Sockets Layer |
| CA | Certificate Authority |
| RESP | Redis Serialization Protocol |

# Contents

# List of Tables

# List of Figures

# 1 Introduction

Electronic displays that show digital content are indeed replacing traditional notice boards, billboards and so on. Nowadays, these displays are connected directly to the Internet and serve content from the cloud. This has opened up new possibilities for remote access and management of these displays. The proliferation of pervasive displays has also increased over time due to technological advancements in their hardware. Moreover, there are large and multi-screen displays that support rich user interactions.

## 1.1 Research Scope and Goals

Many of the digital displays are deployed in public areas and it is difficult to estimate how many people viewed the content. Besides, it is challenging to attract users to view and interact with the content. With interaction types based on touch, gesture, and mobile phones, it is now possible to support interactive applications. As the number of these displays increase, it is critical to provide user-friendly and efficient solutions for managing them remotely from the cloud.

We investigated existing solutions for remote management of pervasive displays and found that most of them require installation of native software. A recent work adopted web-based solution to manage the displays and also supported user interactions with their mobile phones [53]. However, mobile devices are energy constrained: performing heavy-duty tasks on these devices consumes a large amount of resources. There are practical solutions such as code offloading, where complex computing tasks are performed in the cloud [5, 32]. An example is given by an image recognition software where complex calculations run on the cloud while the client with a light-weight software displays the returned results from the cloud. Alternate solution is to get remote access to the display using thin client software such as Virtual Network Computing (VNC), Remote Desktop Protocol (RDP) solutions [55]. This type of software connects to a remote desktop server and provides access to server applications. The client application gives a desktop user interface to the client and its inputs are sent to the server. However, these solutions are mainly designed for desktop computers and have performance issues when used in mobile devices. Also, these solutions are cumbersome to use on devices such as smartphones and tablets. There are proprietary solutions such as SmartVNC [67] for mobile devices but they are not freely available. This thesis instead, focuses on providing an efficient solution in terms of network utilization and delay for remote management of displays. The proposed solution is based on an extensive literature survey conducted to characterize the research challenges in the management segment of the pervasive displays. This thesis considers web based solutions due to the various benefits that they offer: device and platform independence, ease of use, the need for browser support only and no additional hardware to play media content.

## 1.2 Contributions

This thesis leverages modern web technologies specifically – HTML5, WebRTC and WebSocket – for remote management of pervasive displays. This thesis extends the recent research work by Oat et al. [42] that provides a web based solution for controlling the content with the users mobile device. Specifically, we build a web-based solution for remote management of pervasive displays. We employ three state-of-the-art JavaScript libraries for real-time collaborative web applications and implement a representative HTML5 application for digital signage. We then evaluate the performance of the underlying approach in terms of Round Trip Time (RTT), overhead and page loading time. We conduct different experiments and measure these metrics in different network conditions.

This thesis attempts to provide highly efficient solutions for remote management of displays. Our work is helpful for future research on creating and scheduling multiple contents to the displays. We also suggest few real-time frameworks not used in this thesis that could be considered for future research.

## 1.3 Structure of the thesis

This thesis is organised as follows. Chapter 2 briefly introduces the history of pervasive displays and then presents their software architecture, along with the major techniques used for remote management. Chapter 3 overviews traditional bi-directional communication techniques for web applications and their limitations; it then introduces current technologies that can be adopted. Chapter 4 describes the setup of our experimental testbed and the results of the experimental evaluation conducted. Finally, Chapter 5 concludes the thesis and discusses possible future work.

# 2 Pervasive Displays

Pervasive displays are digital displays that show different contents such as text, video, audio and so on. These displays are deployed in public, semi-public and private spaces such as shopping malls, railway stations, offices and so on. Pervasive displays are also called ubiquitous displays or digital signage. In this thesis we use, these terms interchangeably, unless otherwise stated. These displays gained popularity because of different features that they provide including push-based distribution (for example advertisements, emergency announcements), context-specific content (for example web advertising), multimedia content (for example images, videos), and are easy to upgrade. Advertising is the most important domain for the use of display networks, as these displays are majorly supported by advertisements [7]. These signage systems are being transitioned from showing content only systems to systems that support communication and interaction. Technical work in this type of display networks is often carried out by the graphics (Human Computer Interaction) and ubiquitous computing communities. In this chapter we briefly introduce, the evolution of pervasive displays, various reasons and techniques used over the years for remote management of pervasive displays.

## 2.1 Evolution of Pervasive Displays

Research in the field of Pervasive displays and digital signage started almost 30 years back. Since then, tremendous innovations took place in this field. The digital displays were first put into public use in the form of "media links" during the 1980s. The media links are created by joining both audio and video links [7]. In the 1990s various methods were introduced to install information displays in the public environments. One such method is Flexible Ubiquitous Monitor Project (FLUMP) [21] that used traditional LCD's to display multimedia information. Also, wearable devices were explored for displaying information during that period. In the early 2000s various methods were explored to use small digital displays for displaying information in workplaces (e.g., displays as doorplates) [7]. Also, during that period effective use of the pervasive displays in improving awareness and creating a sense of community at the workplace is investigated [7]. The capabilities of these device are further extended to support various features such as live video conversations, display items of relevant information such as live news, weather feeds and so on. Many other solutions were developed to provide awareness, and to promote social interaction. In the late 2000s deployment of pervasive displays in public places such as city centers and university campuses has significantly increased.

The hardware features for public signage displays evolved from the early split-flap displays to the recent development of ultra high definition or 4k resolution displays, and wearable devices like Google Glass [7]. In split-flap displays a series of flaps are rotated mechanically to form the display as shown in Figure 1a. In today's world, there are also video walls (i.e., multi-monitor displays such as the one shown in Figure 1b) that are designed to overcome the size limitation of LCD displays.

Prior to web technologies, native software was installed on display systems to

serve various purposes. With the evolution of Internet technologies modern browsers (thin-clients) have the ability to play high-quality videos and also support real-time communication. Therefore many signage systems these days use web technologies to distribute and manage the display content. Also, with thin-clients a wide-range of display systems can access the full range of digital signage functionality. Modern pervasive displays are equipped with the latest display and sensing technologies, and can connect to the Internet over a wireless connection. Also, these displays can be managed and serve content from servers deployed in the cloud [7, 53]. Modern displays also support different user input types such as touch, gesture and mobile device-based interactions [7]. This has enabled the content designers and display manufacturers to support wide range of features [7].

Despite the innovation in technologies and display systems, there are many design challenges involved in creating and distributing the content that add real value to the viewer. The key challenge here is to identify the capabilities of the installed pervasive displays in public areas [7].


(a) Traditional Split-flap displays [3]

(b) Video Wall displays [26]

Figure 1: Display types

## 2.2 Reasons and Requirements to Manage Pervasive Displays

Remote management of pervasive displays is necessary for various important reasons such as:

- Manual monitoring and service restoration during faults is cumbersome.
- Monitoring the display state changes is also a challenging task that is to be handled by the management part of the signage architecture [7].
- In the case of open display networks [8] the concept of display app-stores allows both users and display owners to purchase the applications from a central repository and expects it to be displayed on the screen. This requires management of interfaces such as those between application server and the central repository for proper scheduling.
- Securing these displays is critical for various reasons particularly when the displays support user interactions. For example a user is susceptible to evil twin attacks where the attacker attracts the user to connect to rogue access

point (i.e., same SSID of the original one) and access the phishing pages and steal sensitive data [11].

- Furthermore, irrespective of the location of the digital displays, the content must be distributed to these displays almost instantaneously.

These displays also require an easy and non-technical installation and configuration process. They must show the appropriate content that is targeted and needs proper scheduling of the content. The management component must be designed carefully to handle these challenges and several techniques were proposed over the years to solve some of the challenges.

## 2.3 Software Architecture of Pervasive Display Networks

The architecture for the digital signage systems can become quite complex based on the purpose to be solved. However, we base our study on the basic architecture for signage systems [7, 53] as shown in Figure 2. There are mainly four components in the software architecture of pervasive displays networks which are content creation, scheduling and management, display component and interactive devices such as mobile phones.

In the content creation component, the software and tools used are generic, thus not specific to signage systems. However, most signage systems provide tools with simple interface for creating content. These tools can be used for creating different types of content such as images, videos, audio, animations and textual information.

The main research area of the pervasive displays is scheduling and management and Human Computer Interaction (HCI). This scheduling and management component serves the back-end for the signage system architecture and takes care of distributing the content to the appropriate screen parts of the displays. Modern commercial signage systems use cloud services for managing the content.

- To achieve content scheduling in non-interactive signage systems, signage operators usually create display groups and content playlists. The scheduling and management part takes care of mapping and scheduling of playlists to the display groups. In interactive signage systems the functionality is complex due to the factors to be considered such as user interaction, content upload by third parties and so on. This segment has many challenges to fulfill due to a wide range of scheduling requirements such as displaying content at specified times, content rescheduling based on user interaction and displaying the same content for a certain number of times.

- From an architectural perspective, it is beneficial to separate the management of pervasive displays from content scheduling and distribution. The display management component handles various administrative functions such as remote monitoring of displays, controlling the power states of the device and handling software updates.

The display component (i.e., displays) shows the content served by the cloud servers. The other features such as sensing and interactions are also supported by the displays.

The device based interactions are often done using mobile devices. Mobile devices act as a gateway to the displays in the local physical environment. These mobile devices can be used to control/manage the content on pervasive displays.

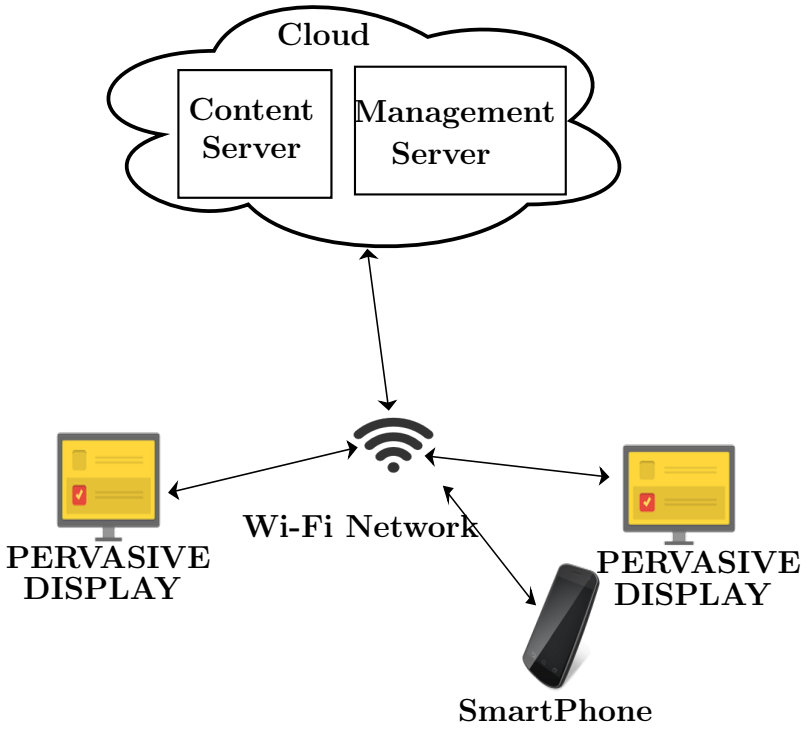



Figure 2: Basic Digital Signage System Architecture in the recent times

## 2.4 Techniques

There are many ways to manage and distribute the content securely to the appropriate displays. With the recent advancements in the ubiquitous computing, it is possible to manage and schedule different types of applications such as real-time interactive applications, traditional digital signage applications that are often non-interactive and both these types of applications concurrently [13]. Some of the projects that support both the interactive and user-contributed content are Instant Places, Screens In The Wild (project aimed to enhance the digital displays design to benefit public life), UBI-hotspots (large multipurpose public interactive displays deployed in Oulu), and e-campus (digital displays connected in a network and used for regular signage within a campus) [7, 13].

There are various types of techniques introduced over time and are categorized. These techniques are described next.

### 2.4.1 Mobile device-based

Mobile based interactions are quite popular for some time and is an effective way to attract users by allowing them to interact with the pervasive displays. There are different ways for a mobile device to communicate with the nearby digital display. Earlier techniques used Bluetooth for the communication between the mobile device and the display located in the physical environment. Later on, due to technical advancements mobile phone applications were developed to send user inputs to the applications deployed in cloud for controlling the content on nearby pervasive displays. Near Field Communication (NFC) tags of the devices are also used to establish communication between mobile phones and the nearby pervasive displays. Latest techniques use Quick Response (QR) codes and RFID for accessing information from the pervasive displays [7]. Oat et al. [42] provide a solution that allows users to interact with the pervasive screens using their mobile phones. The proposed solution is a complete web-based solution that leverages latest HTML5 technologies. It also provides a solution to establish a secure connection between pervasive displays and users mobile device.

Sethi et al. [53] propose a technique for secure management of cloud connected digital displays, by introducing a new user-assisted protocol for wireless displays to establish a secure connection to both the cloud service and wireless access network. The proposed architecture have separate content and management servers. Connection is initiated when the user's mobile device scans the QR code that is displayed on the screen, and then after the one-step configuration process, the content is showed on the display.

Erbad et al. [16] proposed a messaging broker (i.e., a middleware) for interactive large screen display applications which supports mobile device based interactions. The proposed middleware solution uses abstractions such as channels (i.e., topic in publish/subscribe pattern), events, state, services and content. However the solution is built using either outdated or proprietary protocol.

Google [41] proposes an end-to-end framework that uses mobile phone camera to project the content from the user's mobile device to the target pervasive displays or personal computer. The proposed solution uses QR (Quick Response) code to identify the pervasive display. The content projection to the target display takes place in three steps. The user first scans the QR code of the target display, then the barcode turns to checkerboard marker and moves until the user identifies the target position, and then the content will be projected to the target position. The solution used web-based architecture for deployment and accessibility to the projector service.

### 2.4.2 Multi-content and Multi-application Support

Strohbach et al. [62] propose context management framework (CMF) that is able to evolve with customer needs and new technologies. They developed this framework to enable bidirectional communication between advertisers and consumers, which helps the advertisers to increase their campaign efficiency and also add value to the consumers. Their framework consists of context agents that provide information to

the applications with the data received from sensor sources.

Linden et al. [33] propose a web based framework for dynamic partitioning of the display screens into several virtual screens to accommodate multiple concurrent web applications both in spatial and temporal dimensions. The proposed architecture is a scalable and open architecture that allows integration of third party web applications into UBI-hotspots. The solution used resource manager to control the allocation of screen estate of the displays to the concurrent web applications and the layout manager to control the actual layout of the screen that is partitioned into virtual screens to accommodate different web applications.

Johanson et al. [28] propose a "multibrowsing" framework that allows free movement of the web based content across ubiquitous displays. The proposed solution leveraged web technologies to build the framework. The framework allows three different types of roles that can be given to the displays. Based on display's role they may have to install or run custom plug-in or service.

### 2.4.3 Content Distribution and Scheduling

Clinch et al. [6] provide a solution to design application stores for ubiquitous displays, to distribute third-party applications in open display networks [8]. The proposed solution concentrated on scheduling and distributing the signage content to the pervasive screens using an application store. The authors also reported the design considerations and the limitations of using such an approach. The User Interface (UI) design in the proposed solution claimed to meet the requirements of application developers and display owners. The proposed solution solves the purposes such as, it allows application developers or content providers to upload their content, it allows display owners to view, purchase, and organise applications and it also allows the display owners to manage their physical hardware.

There are software solutions such as Scalable Adaptive Graphics Environment (SAGE) by Jeong et al. [27] middleware, developed to display relevant content on multi-monitor displays. The SAGE middleware supports streaming real-time data, High Definition (HD) videos, and high resolution graphics to the scalable display walls.

### 2.4.4 Human Computer Interaction

Kuikkaniemi et al. [31] employed large interactive public displays to explore the presentation behaviour in walk-up-and-display scenarios. Specifically, a specialized system was designed to provide presentation features in addition to the information browsing feature. The work aims at providing a near real-time experience for content management (i.e., publishing and subscribing the content), although the details are not elaborated.

Heikkinen et al. [24] provided a description on the remote monitoring and management tools developed for the maintenance of UBI-hotspots. The "Nagios" remote monitoring tool they used is able to detect component and system level issues. However it could not detect issues in the web-based user interface. To monitor the

user interface issues, “Happy page” software is used to fetch the period screen capture updates from the hotspots. Further Remote Desktop Connection (RDC) and Virtual Network Computing (VNC) are used for diagnosing the faults.

### 2.4.5 Web-based or native software based solutions

Heikkinen et al. [25] proposed an event-based communication middleware for a network of large multipurpose displays (i.e., UBI-hotspots). The proposed solution uses RabbitMQ message broker, a messaging middleware that supports publish/subscribe pattern and Remote Procedure Call (RPC) to support multiple messaging patterns. However the message broker does not support transfer of raw streams of media intensive data.

Olberding et al. [43] propose a solution to various challenges posed by CloudDrops, a set of many tiny interactive stamp-sized displays and each display shows a bit of digital information. The solution provides a way to map the content to displays, manage the screen content, and handle user interactions. The proposed solution supports users to view and interact with three classes of content, which are document related content, people related content and content related to places. The content is uploaded to the cloud drops in three different ways, one way is by placing CloudDrops near to the devices (PC, Laptop, or MAC) and the changes in the user selected web page snippets are reflected on the CloudDrops, the other way is by attaching CloudDrops to locations such as walls, doors, and desks, and finally by grouping the CloudDrops.

Despite all the various techniques mentioned in this chapter, there is no efficient web based solution for remote management of ubiquitous displays. We aim to provide a complete web based solution that is effective in terms of performance. Such a solution is discussed in Chapter 5.

# 3 Technologies

In this chapter, we introduce the technologies that are used in our proposed architecture that is detailed in Chapter 4. Firstly we discuss about the evolution of bi-directional communication techniques used in the web, followed by HTML5 technologies. We then discuss different types of solutions for desktop sharing. Specifically, we introduce the concepts of Operational Transform (OT) and Network Time Protocol (NTP) to understand the functioning of the relevant libraries that we used. We also discuss Publish/Subscribe systems. Finally, we detail state-of-the-art JavaScript libraries that we researched on.

## 3.1 Legacy Bi-directional Communication Techniques for web

In the conventional client/server network architecture, the client is either a device or an application and the server is a machine that runs software locally to share their resources with the clients. Often clients start the communication with servers, however this approach is not suitable for modern real-time applications. Since the HTTP protocol is a request/response protocol where the client sends a request to the server and expects a response from the server, conventional web pages must be reloaded, when there is an updated content and as a consequence this creates a large overhead. Due to this limitation several techniques were developed to establish asynchronous communication where the responses for corresponding requests are received without reloading the page. These techniques open multiple HTTP connections to establish bi-directional communication. There are techniques such as polling, long polling and streaming over Asynchronous JavaScript and XML (Ajax), which are executed at the client side for retrieving data asynchronously from the web server.

Among them Ajax is a technique used by the client to fetch data asynchronously from the server by sending a HTTP request to the server.

With the polling technique, the client sends an AJAX script with the request and expects to be served with the data from the server, asynchronously. However, when the server could not serve the content due to unavailability, the client starts polling the server repeatedly until the content is served.

With the long polling technique, the server maintains timeouts and during the timeout period the server holds the client's request in its message queue until the content is available. On content availability the server sends a response to the client.

With the HTTP streaming technique, the server uses a persistent connection with the client and pushes updates to the client when available. Due to the buffer mechanism used for streaming, this approach increases the delay in delivering messages, thus it is not suitable for resource constrained devices. Also, in persistent connections it is not efficient to allow the client and server to stay connected for a long time due to various factors such as the time-outs values set by the servers, that is used to kill the idle connections, limiting the number of simultaneous connections initiated by the clients to the servers to a maximum of two.
The consequences of both polling techniques are [20]:

- They increase the workload of the server for keeping the requests open.

- The HTTP header in each request leads to high overhead in the transport protocols because of the number of requests made.
- It increases the frequency of polling the webserver.
- A mapping from the outgoing connections to the incoming connections has to be maintained on the client side.
- The number of different underlying TCP connections increases on the server side for upstream and downstream exchange of messages with the client.

## 3.2 BrowserChannel

BrowserChannel is a protocol developed by Google to provide bi-directional communication [29]. It does long polling and does not support WebSockets and cross domain requests. Also, it does not support Remote Procedure Call (RPC). The following features are provided by BrowserChannel [29]:

- Messages arrive in order.
- Messages are never received by the server after the connection is closed.
- It works on all major browsers.
- It can send messages even before the connection to the client is established, as opposed to WebSockets. This saves an extra round-trip if messages are sent along with the connection request.
- Messages are automatically converted to JSON messages through Google's JSON encoder.

## 3.3 HTML5

HTML5 is the fifth revision of the HyperText Markup Language that is used to create hypertext documents. HTML5 aims at defining a single markup language that can be written in HTML or XHTML. HTML5 specification includes the HTML4, XHTML1, and DOM2 HTML specifications [2]. HTML5 includes several technologies that allows to create diverse and powerful web-based applications. The following are some of the powerful HTML5 technologies:

- It provides improved semantics by adding more meaningful elements that can be used based on the content. Specifically, it adds several new elements like audio, video, mark, figure, figcaption, data, meter, time, output and so on. It added improvements in HTML forms and iframe. It added MathML application to embed mathematical formulas directly.

- It adds new connectivity features. These include WebSockets that creates a permanent connection between user agent and the server, Server-sent events (for the server to send events to the client) and WebRTC (for real-time communication such as video calling between peers over the web without the need for external plugins).
- It allows webpages to store data locally on the client side. It also supports online and offline events.
- It allows to embed and modify multimedia elements such as audio and video. It is also capable of using the camera Application Programming Interface (API).
- It added canvas element that allows to do 2D/3D graphics on the Web. WebGL API allows to add 3D graphics to the web.
- It brings in major performance improvements and better integration by introducing Web Workers (allows scripts to run in background threads).
- It is capable of using various input and output devices in terms of touch events, geolocation, device orientation detection, pointers and so on.
- It adds several new features for styling that allows to create complex styling and the newest version of cascading style sheets is called CSS3.

## 3.4 WebSocket

WebSocket [20] is a new advanced and independent protocol used to establish a full duplex, bidirectional communication over a single TCP connection. WebSocket runs over HTTP ports 80, 443 and also support HTTP proxies. The WebSocket protocol uses two new URI schemes 'WS' and 'WSS' for non-secure and secure connections respectively [20]. In the handshake procedure, WebSocket connection is established by sending an upgrade request using the upgrade request header supported in the HTTP/1.1 version. During the connection establishment, if there is a proxy server that is used by the client, then HTTP connect method is used to setup a persistent tunnel. Also, client can request the server for using WebSocket subprotocols by including the header field "sec-websocket-protocol". After the connection is established, WebSocket supports exchange of message-oriented text and binary data frames between client and server as long as the connection is available [70]. The WebSocket connection establishment is shown in Figure 3. To terminate the WebSocket connection, endpoint must use a clean method to close the underlying TCP connection and TLS session. To start the WebSocket closing handshake, the endpoint sends the status code, reason to close and sends a close control frame. On sending and receiving the close control frame, the endpoint closes the WebSocket connection [70].
WebSocket API is event-driven and greatly improves performance for real-time and event-based communications. Moreover, WebSocket supports sub-protocols that is useful to build modular and reusable components. The limitations in establishing WebSocket connections directly as per the research done by the engine.io team [60]

are WebSocket traffic is blocked by many of the corporate proxies, the antivirus and firewall applications used for personal computers block the WebSocket traffic, and a few widely used cloud platforms such as Heroku prefer long polling to WebSocket.

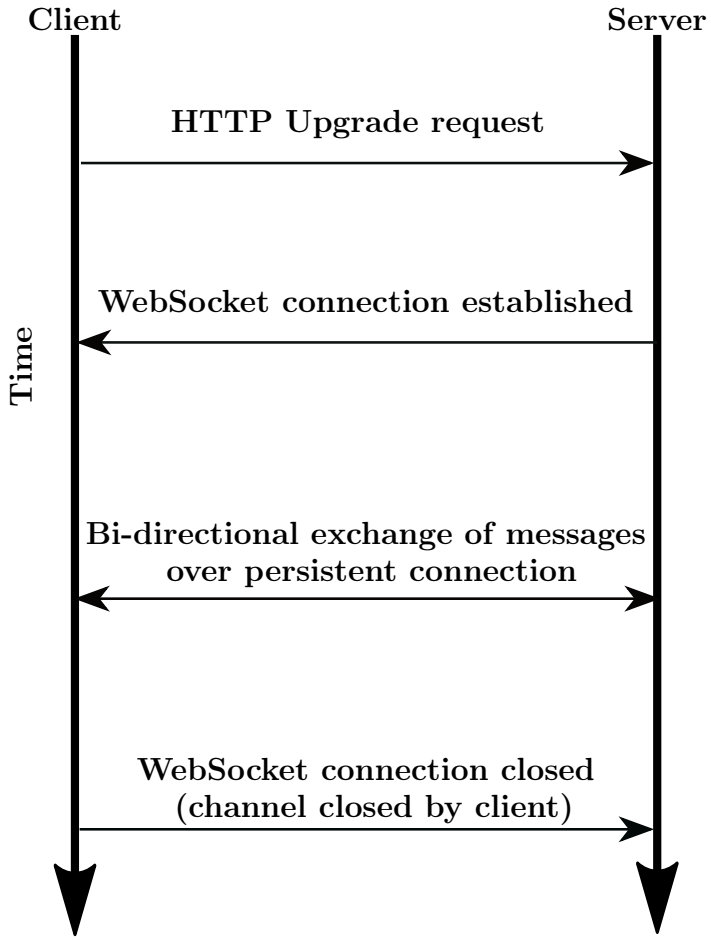



Figure 3: Websocket Connection Establishment

## 3.5 WebRTC

WebRTC [1] is an API standard that provides plugin-free browser-to-browser, real-time, audio, video, and data communication. The WebRTC stack provides three APIs namely, Mediastream, RTCpeerconnection and RTCdatachannel. There is also a proposed API [52, 71] to capture browser tab content as a mediastream that can be sent over WebRTC.

## 3.6 Desktop Sharing Solutions

Desktop sharing solutions employ the common client/server model. In these cases a thin client software (also called remote desktop software) is installed to access a computer's (server) desktop environment remotely from another computer (client)

over a network. A client machine accessing the remote computer needs to authenticate itself first, the entire screen content is then sent to the client machine. The remote desktop software captures client computer's mouse and keyboard inputs and sends them to remote computer (server). The remote computer (server) calculates the changed screen area for the corresponding inputs and then responds by sending screen updates to the client machine. The operations involved in remote desktop sharing are shown in Figure 4.

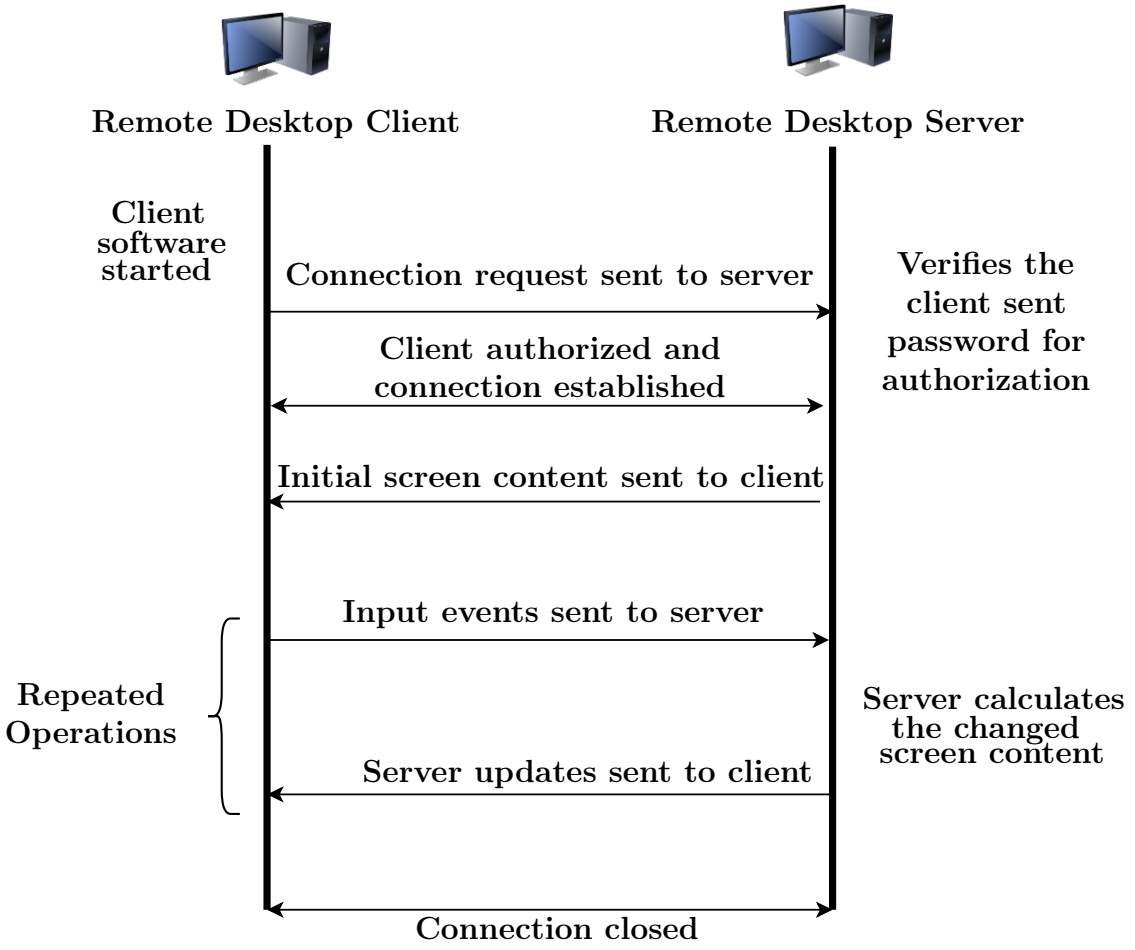



Figure 4: Remote Desktop Sharing Operations

### 3.6.1 Virtual Network Computing

Virtual Network Computing (VNC) is one of the remote desktop sharing solutions, which uses Remote Frame Buffer (RFB) protocol for its operation. VNC is a platform independent solution. Multiple VNC clients can connect simultaneously to the VNC server.

The RFB protocol works at the frame buffer level and supports all windowing systems and applications [61]. RFB opens a long-lived process on the server to

maintain the state of the framebuffer and make client stateless. Typically, RFB clients connect to the server on default TCP port 5900, manipulate the framebuffer for a period of time and then disconnect [61].

In RFB, the display protocol takes care of drawing rectangles of pixel data at particular coordinates and allows different encodings for the pixel data. These sequence of rectangles together form a framebuffer update and is sent by server to the client upon request. The input protocol takes care of capturing the input events and sending them to the server. The pixel data is sent in a particular format (i.e., the representation of individual colours by pixel values) and encoding (i.e., the way to send rectangle of pixel data) negotiated by both client and the server during the initial handshake [61].

RFB operates on any reliable protocol such as TCP/IP. There are three phases in RFB protocol to establish connection and exchange data between client and server as shown in Figure 5. It starts with the handshake phase in which the server first sends a ProtocolVersion message consisting of 12 bytes to the client. The client then responds to the server, with a message specifying the ProtocolVersion to be used. These steps complete the ProtocolVersion negotiation phase. The client and server then negotiate on security to be used, by exchanging security type (no authentication or VNC authentication) messages. In VNC authentication server first sends a random 16-byte challenge to the client and client responds to it by sending a 16 byte encrypted message [61]. The second phase is the initialisation phase during which the client first sends ClientInit message, specifying if the server should share the screen, by leaving all the clients connected or by disconnecting all clients. The server responds by sending ServerInit message specifying dimensions of the server's frame buffer, pixel format and the device name. In the final phase normal protocol interaction takes place between the client and the server [61].

RFB supports multiple encoding types, any input device that can be mapped to keyboard and a pointing device, and various extensions for better integration with the remote server [61].

### 3.6.2 x11vnc server

x11vnc server is an implementation of VNC servers and allows a computer to be viewed and controlled remotely and works with any VNC client viewer. This software polls the server's (i.e., machine on which x11vnc is installed) frame buffer for changes. The remote computer (client) can access the desktop environment of the server over the local network or Internet. This software also supports polling non-x11 frame buffer devices such as Linux terminal, webcam etc. It contains built-in SSL/TLS encryption. It also supports VeNCrypt security type. Also x11vnc has client-side caching that is implemented using brute force.

In this solution the polling algorithm reads data of 32 pixels at a time vertically and when it reaches the bottom it starts again from top using a different offset. With this approach it reads data 32 times faster compared to sequential approach of reading pixel by pixel. The process of reading is made much faster by using x11 DAMAGE extension, which gives the position to focus polling.

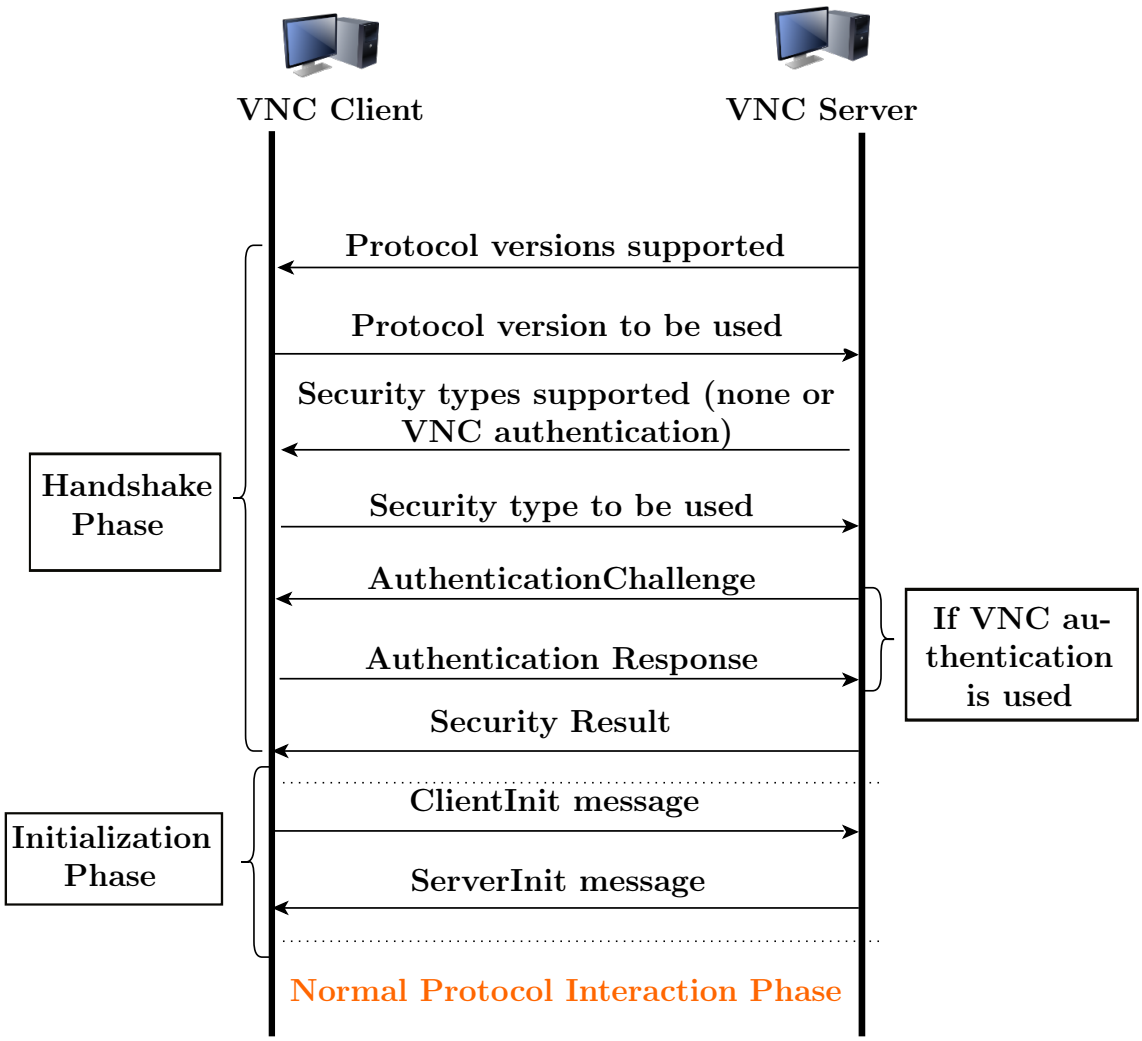



Figure 5: RFB protocol stages

### 3.6.3 Browser based VNC client solution

noVNC is a cross-browser platform, HTML5 based desktop sharing solution. It uses HTML5 canvas, WebSocket and typed arrays. This solution provides a python script (websockify) that is used as proxy and allows the actual HTML5 page to connect to the TCP socket proxy using WebSocket, and the proxy communicates with the VNC server over VNC protocol. The proxy is used if the vncserver variant does not support WebSocket connections. The following features are supported [37]:

- Raw, copyrect, rre, hextile, tight, and tightPNG VNC encodings.
- WebSocket TLS/SSL encryption.
- 24 bit true color and 8 bit color depths.
- Desktop re-size notification and pseudo-encoding.

- Local and remote cursor.
- Clipboard copy and paste features.
- Clipping or scrolling modes for large remote screens.

## 3.7 Network Time Protocol

Network Time Protocol (NTP) is used to synchronize the time of the computer systems connected to the Internet [38]. NTP maintains the time with an accuracy of tens of milliseconds over the public Internet, and achieves an accuracy within one millisecond in local area networks. There can be errors of 100ms or more during asymmetric routes and network congestion [38]. In NTP implementations timestamp messages are exchanged over User Datagram Protocol (UDP) port 123. NTP does not provide information on daylight savings.

NTP client synchronizes its clock with that of the server by polling three or more servers on diverse networks and computes round-trip delay and offset times.

- The round-trip delay is calculated by taking the difference between the elapsed time on the client side for sending a request and receiving response, and the waiting time of the server before sending response [38].
- The offset is calculated by taking the average of the time difference between the time the client sent the request and the time when the server received the client's request, as well as the time difference between the time the server sent the response and the time when the client received the server's response [38].

The Simple Network Time Protocol (SNTP) is a simpler version of NTP with less complexity. It does not store the state for a long period of time and is used in applications where the accuracy in timing is not critical.

## 3.8 Publish/Subscribe systems

The publish/subscribe messaging pattern is an approach to deliver information from publishers (i.e., devices/applications sending updates) to subscribers (i.e., devices/applications receiving updates) in an efficient and timely manner. This is achieved by first detecting the events (i.e., discrete state transitions) by the publisher and then delivering them to active subscribers in an asynchronous fashion [63].

Real-time communication systems need an event-driven notification system such as publish/subscribe, a service that enables to get real-time notifications for only particular events [63]. There are different types of publish/subscribe solutions used for internet applications such as, messages can be sent directly from publisher to subscribers through channels as shown in the Figure 6, through message brokers in which messages are first sent to the message broker which then takes care of delivering those messages to subscribers as shown in Figure 7. It is also possible to increase the scalability of the publish/subscribe network architecture by adding more message brokers in the network [63] to deliver messages as shown in Figure 8, which

is the most common publish/subscribe solution provided in distributed environment. The latest technologies that support Publish/Subscribe pattern are WebSocket Application Messaging Protocol (WAMP), MQ Telemetry Transport (MQTT), Pubsubhubhub protocol. It uses JSON as its message serialization format.

- WAMP [68] is an open standard WebSocket sub protocol that provides structured messaging by implementing application level messaging patterns: Publish/Subscribe and Remote Procedure Calls (RPC).
- MQTT [17] is a lightweight publish/subscribe messaging pattern based protocol designed specifically for resource constrained devices. It is designed to minimise network bandwidth and to guarentee reliable delivery of messages. It is widely used in "Internet-of-Things" and "Machine-to-Machine" applications.
- Pubsubhubhub [22] is an open protocol that provides publish/subscribe based communication. It provides a solution to subscribe, unsubscribe and receive updates from a web resource. Publishers include references for hub in their content. Subscribers then access the URL and checks for the hub references in the received response. If the references are found they subscribe to that resource URL and then receive updates from those resources.

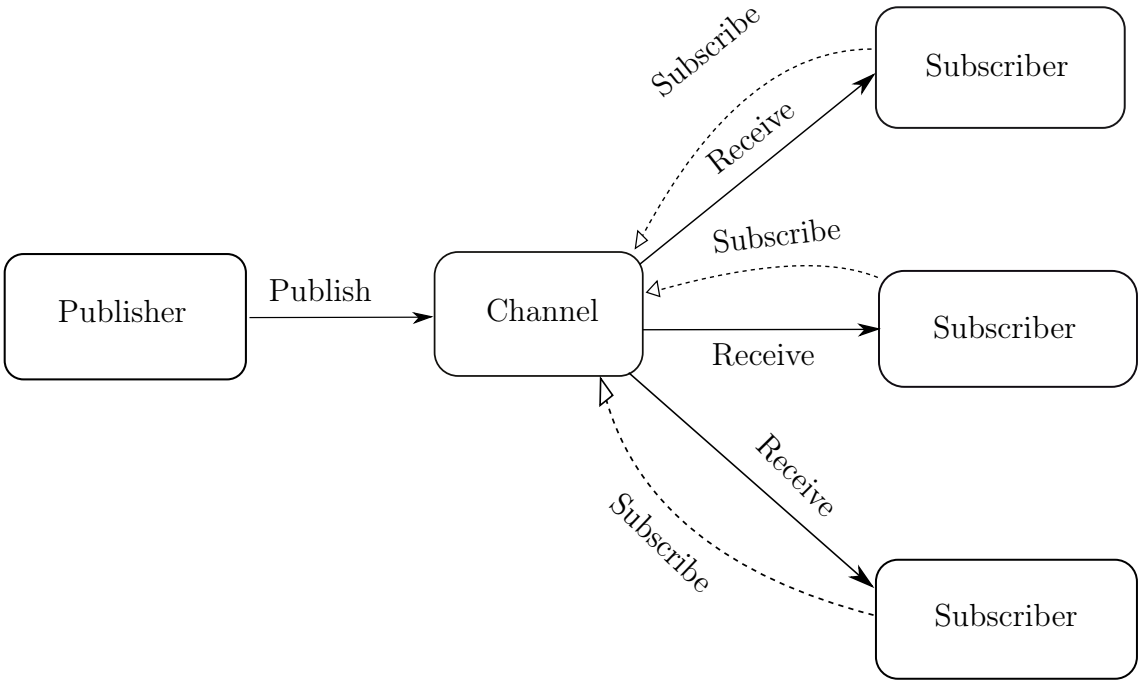



Figure 6: Publish/Subscribe pattern

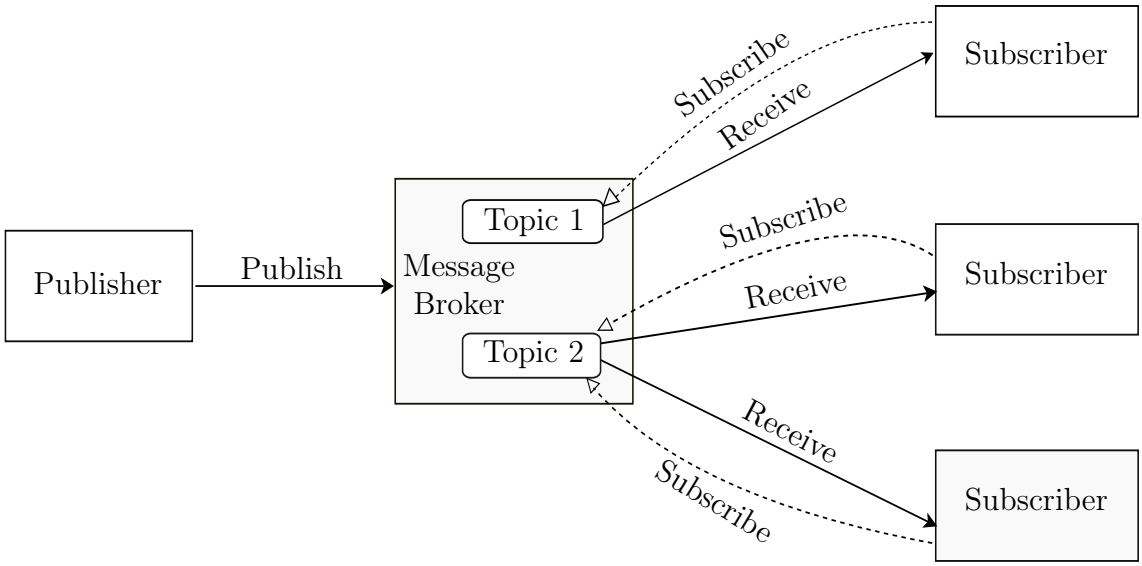



Figure 7: Publish/Subscribe pattern with a message broker

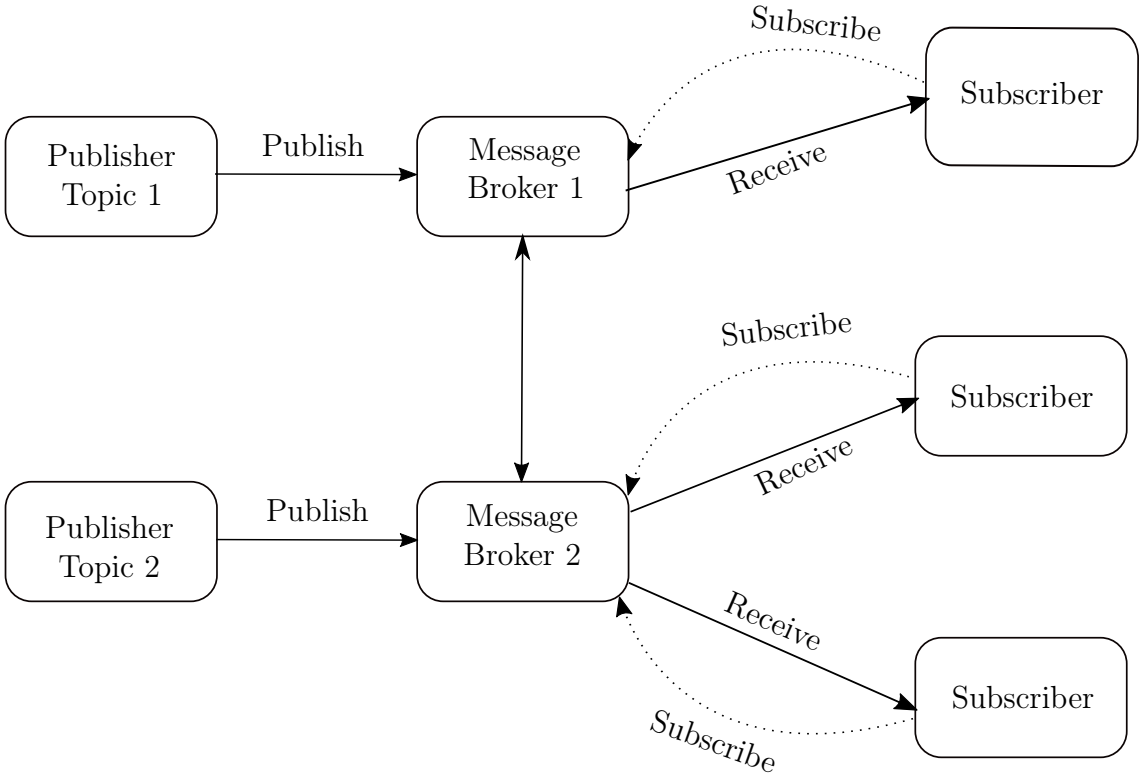



Figure 8: Publish/Subscribe pattern with a network of message brokers

## 3.9 JavaScript Libraries

To share and control the content on remote displays we have VNC solution. Even though we have a browser based VNC client solution, we need additional software and intermediaries to setup the VNC server. We aim to provide a complete web based solution to remotely control the content on digital displays. Also, we aim to

provide a solution with better performance. Therefore, we did an extensive literature survey to find the suitable libraries to implement our solution. The features provided by these JavaScript libraries are detailed next.

## 3.10 Faye

Faye [18] is a publish/subscribe messaging system that uses Bayeux protocol [51] (i.e., a protocol used to transport asynchronous messages over HTTP with low latency) to exchange messages asynchronously. It supports messaging server for both Node.js and Ruby. Faye-websocket [19] is a WebSocket client/server implementation for Node.js built on top of Faye project. It also support event-source connections (one way connections and allows server to push the data to the client). It is compliant with the existing WebSocket standards.

## 3.11 Mutation Summary

In the old standard [73] (i.e., DOM level 3 Events specification) mutation events were used to notify any changes made to the structure of a document including attributes, name and text modifications. Mutation events is a synchronous operation and its design has significant performance issues such as, they fire for every single change, they are slow due to event propagation, they are the source of crashes on many user agents. Also they have improper cross browser support and due to these reasons mutation events are replaced by mutation observers in DOM Level 4 standards.

Mutation Observers use callback functions, which are used to notify multiple changes in the DOM at once. Mutation observers is an asynchronous operation and in this approach, nodes in a document are observed for changes and the corresponding callback functions are triggered only after all the changes are applied to the DOM [12]. After this, the triggered callback function will have all the DOM changes. Mutation observers are supported by all modern browsers [12].
Mutation observers have the limitations [45, 12] that is they do not support detecting CSS style changes such as hover state, they do not support logging timestamp details of mutations in change records, and the active internal state of the form elements such as the value of textarea element is not identified accurately.

The Mutation summary library [48] is built on top of the latest DOM mutation observers API and is an efficient and reliable library. Mutation summary takes all the changes notified by the mutation observers, computes the net changes that took place and then delivers those changes to the callbacks. This library provides the following features [48]:

- It supports four different types of DOM changes that can be observed throughout the sub-tree, and the changes are attribute changes, element changes includes only a simple subset of CSS, character data changes, and all changes.

- The time and memory it takes is proportional to the number of changes that took place; as these changes typically involve a few nodes. It performs faster in detecting changes.

- It ignores changes made during the callback.
- It can handle complex operations.
- It is supported by all modern web browsers.

The limitations of this library are, it does not support pseudo-element matching, it does not notify the switching of DOM transient states, it does not notify accurately the internal state changes of the embedded iframe players.

## 3.12 Redis

Redis [50] is an in-memory data structure store which can be used as a database, cache and message broker. The Redis Publish/Subscribe messaging system consists of PUBLISH, SUBSCRIBE and UNSUBSCRIBE mechanisms. The transfer of messages from publisher to subscribers takes place through channels. The Redis client-server communication takes place over a specially designed Redis Serialization Protocol (RESP). Redis supports pipelining and the client can send multiple commands to the server without waiting for response [50]. The RESP protocol is used only with stream oriented connections, such as TCP connections, and is capable of serializing data types such as integers, strings, bulk strings, errors and arrays. Redis client sends requests to the Redis server as an array of bulk strings representing the arguments of the command and server replies with one of the RESP data type, specific to the command [50].

## 3.13 livedb

Livedb [54] is a wrapper for real-time databases and the current API version has binding for mongodb database only. Livedb also has in-memory database backend to store all the documents and operations in memory forever. The data model of livedb has three properties, namely version, type (OT type) and data (i.e., document data).

Livedb requires three important inputs to operate appropriately. A database to store actual documents (snapshots), an operations log to store operations and a livedb driver (in-process or Redis) for managing communication in a multi-server architecture. Livedb client can be created using either an options object or a database backend and if database backend is chosen both operations log and snapshots can be stored in the database. The Redis server has to be used for multi-server architecture and two redis clients are to be created to handle commands and pubsub respectively. Livedb supports live queries against the database, projects the real collection of data with only a limited fields for only JSON documents.

## 3.14 Operational Transformation

Operational Transformation (OT) [14] is a technique used to maintain consistency and concurrency control in collaborative software systems. The main idea of the OT technique is to execute transformation of parameters of an editing operation based

on the previously executed concurrent operations. This logic ensures accurate results and maintains document consistency across all sites [44]. Also, the local response time is insensitive to network latencies in OT systems [44]. It provides a range of collaborative functionalities such as group undo, conflict resolution, locking, operation notification and compression, HTML/XML and tree-structured document editing etc [44]. Google Docs [44] and Apache Wave [69] use OT as their core technique.

The OT system structure comprises different components. The most common OT system design separates the high-level transformation control algorithms and low-level transformation functions [44]. The transformation control algorithm layer takes care of identifying the operation to be transformed against the new operation and also the order of transformations. The transformation functions layer determine transformation of operations based on the operation types, positions and other parameters. These transformation functions are invoked by the control algorithms. The responsibilities of the above mentioned layers is determined accurately based on the transformation properties and conditions [44].

## 3.15 ShareJS

ShareJS [23] is a client and server library that can be used to implement real-time concurrent editing web applications. This library is built using operational transformation (OT) algorithm on text and JSON content. The multi-site real-time concurrency is achieved using this OT algorithm. This library works on all major browsers.

ShareJS creates a document object at the server and also maintains the document version. ShareJS generates operations (i.e., mini commits) as and when the user gives the input data to the web application. These operations are applied to the document version. When multiple users are editing the same version of the document, server handles the operations by taking one of the user's operations directly and the rest of them are automatically transformed by the server [23].

The server API exposes three methods namely share.listen(stream) to hand over control of the stream to sharejs, share.rest() returns a connect or express router to expose sharejs REST API, and share.use(method, function(action, callback)...) to intercept requests to the livedb (i.e., a wrapper for real-time databases) backend for access control using sharejs middleware [23]. ShareJS access the database through a livedb client.

The client API exposes two classes, Connection and Doc to handle communication to the sharejs server and store an in-memory copy of the document data including local edits. The Connection class instance is used to create document references in the client [23].

ShareJS allows client server communication over browserchannel, WebSocket or any other transport protocol that guarantees in-order message delivery. This library is able to provide a node object stream to the server to communicate with the client. It also provides a WebSocket like api on the client. The transport used must handle client re-connections. The browserchannel transport takes care of stringifying JSON messages and also supports sending messages while the connection is in progress, but

for other transport mechanisms separate flags have to be enabled to achieve those functionalities.
Document objects in ShareJS follow the Livedb object model, which is a database front-end for concurrent editing systems and has the following features [23]:

- They store data on the client side.
- Client can edit data synchronously and sync it with the server automatically.
- Client can also edit data offline and on reconnect data is sent to the server.

The current version of ShareJS supports Plaintext type OT and JSON type OT. The plaintext OT [64] specification considers document as a large string and edits the string index directly. Operations are list of components, that traverse along the document. Each component could represent one of the following actions, a number N that represents to skip N characters forward, str that represents to insert string at the current position, d:N that represents to delete N characters at the current position in the document. This OT type also supports manipulation of selections. The time complexity of this JSON OT type algorithm is $O(N + M)$.

The JSON OT type can be used to edit arbitrary JSON documents and supports the operations [30] such as insert, delete, move or replace items in a list and objects, numerical addition, string editing, arbitrary subtypes such as text, rich-text and to register them "json0.registerSubtype" is to be called.

## 3.16 Socket.io

Socket.io [56] is a library used for building real-time web applications and is written in JavaScript. It has both client side and server side libraries and enables real-time bidirectional event based communication. It relies on the WebSocket protocol to provide bi-directional communication and polling is used as a fallback option.

Socket.io enables the client to connect to the server over WebSocket at start and if the connection fails it downgrades the connection to XMLHttpRequest (XHR) polling, and again if the connection fails, it is downgraded to flash sockets (used to establish connection from client side flash applications to the server). It also supports socket re-connections [56] and rooms (for example chat rooms) to cluster connections.

Socket.io uses engine.io library for lower level abstraction [39]. The connection procedure in engine.io starts with long polling connection and if the connection fails upgrades it to better transports, which is opposite to the logic that socket.io operates on. Therefore, engine.io provides more predictable results and is more reliable [60].

To minimize the number of TCP connections, socket.io supports assigning different endpoints or paths to the sockets, with the 'namespace' feature and the default namespace being '/' [58]. For each namespace it allows creating multiple arbitrary channels called rooms to which the sockets can join and leave. By default each socket joins the room identified by socket's unique id [58].

In multi-server architecture there is a need to maintain connection between the requests in a session, and the process that originated them, because of the reason that

certain transports such as XHR polling and JSONP polling send multiple requests during the lifetime of the socket connection [56]. To maintain those connections socket.io recommends usage of Nginx server or Node.js cluster (a cluster of Node.js processes). After cluster configuration, for routing messages between multiple nodes socket.io recommends usage of socket.io-redis adapter library that is built on top of redis [59]. For passing messages from non-socket.io processes socket.io recommends usage of socket.io-emitter library in addition to the socket.io-redis library [58].

## 3.17 Derby.js

Derby.js is an Model-View-Controller (MVC) framework designed to implement modern realtime, collaborative web applications. Derby.js uses Racer [10], which is a real-time model synchronization engine for node.js. Racer back-end is built on top of ShareJS and provides a simple model and event interface. Mappings are used to translate racer paths (i.e., unique nested JSON objects) to database collections and documents [10].

Derby.js enable rendering both on server side and client side without duplicating the page rendering code. The technologies used by Derby.js are reactive programming, operational transforms, browserify and mongodb [9]. It also uses isomorphic JavaScript that can execute JavaScript applications both on client and server side. The advantages of this approach are performance optimizations, better maintainability, and more stateful web applications [65].

## 3.18 React.js

React.js [49] is a JavaScript library used to create user interfaces solving the challenges encountered in single page web applications. It handles only the view part in the Model-View-Controller (MVC) framework. It implements one-way reactive data flow. This library can be used alongside AngularJS or any other JavaScript libraries to build real-time interactive applications.

This library uses a technique called virtual DOM (Document Object Model) that abstracts the actual DOM to perform rendering of subtrees based on the state changes. It can do rendering on both client and server side. It achieves better performance by minimising the actual DOM manipulations.

React uses 'diff' algorithm to detect state changes in a tree. The state of the art algorithms, designed to transform the tree structure from one form to another, by generating minimum number of operations have a complexity of $O(n^3)$, where 'n' is the number of nodes. React.js claims that its algorithm has managed to reduce the complexity to $O(n)$ with powerful custom heuristics based on the assumptions that the components generate similar or different trees based on the classes. It is also possible to provide a unique key for elements that is consistent across various renders. React always reconciles trees level-by-level. It takes two diffs into consideration, which are Pair-wise diff and List-wise diff.

In pair-wise diff, the tree diff is calculated based on the diff of two nodes. It identified three different types of node diff such as comparing different node types,

comparing different DOM nodes and custom components diff. In different node types diff both the node types are treated as different sub trees and replaces the first node type (old state) with the second node type (new state changes). In DOM nodes diff, the attributes of both the nodes are compared using key-value object and are updated with the new state changes that took place in linear time. In custom components the logic used is similar to the one used in calculating different node type diff and is required to ensure that both the components are of the same type.

In list-wise diff the reconciliation for child nodes is done by assigning an optional attribute 'key' to the every child node. With this key it identifies state changes such as insertion, deletion, substitution and the complexity for this approach is $O(n)$.

## 3.19 timesync.js

This library [15] is aimed to provide time synchronization in the client/server and peer-to-peer (P2P) networks, as the client's timestamps might not be accurate compared to the server timestamps. In client/server networks the library calculates the offset with the client and server timestamps and in (P2P) networks it calculates the offset by averaging the offset of all peers in the network.

This library uses a simple algorithm in which the client sends the current local timestamp to the server and on receiving the request the server sends the server-time along with the request. The client on receiving response from the server takes the difference of the current time and the sent time and divides the result by two to get the latency. Then time difference between the current time and the server time is added to the latency to calculate the clock time difference.

## 3.20 PhantomJS

PhantomJS [66] is a scriptable, headless browser (i.e., a browser accessed programatically) with a JavaScript API to automate web page interaction. The API enables automated navigation, screen capture, user behaviour and assertions. It is commonly used to continuously run browser-based unit tests in a headless environment. It is based on webkit [47]. PhantomJS does not support CSS 3D transformations, local storage and WebGL. Libraries that use PhantomJS include:

- Pageloadtime [34] which uses PhantomJS and Googlecharts to do performance testing and measuring the page load time of web applications. The output is visually represented using google charts.
- Loadreport [72] which gives a report of the load and speed metrics.
- Phantomas [36], a web based performance metrics collector and monitoring tool.

# 4 Architecture and Evaluation

In this chapter we introduce the system architecture that we used, followed by the details on different types of HTML5-based implementations developed with different JavaScript libraries. We then discuss the experiments conducted to evaluate the performance of the application. Finally, we will provide the metrics used and the results obtained from the experiments.

## 4.1 System Architecture

Our proposed solution is completely web based and leverages HTML5 technologies. There are three components in our architecture, namely the cloud service where the application server is hosted, a mobile phone and a ubiquitous display. The roles of each of the components is as follows:

- The cloud server hosts the application, handles the message exchange between the client (mobile phone) and the pervasive display.
- The mobile phone acts as the controller, meaning that the display content is affected by the input at the device.
- The ubiquitous display takes care of rendering the content on its screen, based on the input messages received from the mobile device.

We deployed our application in the cloud, so that it can be accessed from any network and is more suitable for Network Address Translator (NAT) environments, where the client is behind the private network and can be reached from the public Internet through NAT. The cloud server has two roles: it provides content to both mobile phone and ubiquitous display; it enables bi-directional communication between these devices. The requirements of our proposed architecture is that both the mobile phone and the ubiquitous display must be connected to the Internet and must run a modern web browser that support WebSocket such as Firefox, Chrome, Opera. The system architecture with its different components is shown in Figure 9.

## 4.2 Implementation by scenario

We considered web technologies, specifically HTML5 WebSocket and WebSocket Application Messaging Protocol (WAMP) to provide simple and elegant solution for the digital signage scenarios. We created a basic HTML5 presentation application that contains text and media (i.e., audio, video, images). It supports user interactions such as swipe control to change the slides, HTML5 player controls to control the media. This web application is implemented using three popular state-of-the-art JavaScript libraries mutation-summary, ShareJS, and Socket.io that provide real-time communication. The web application is designed to automate slide changes for a given time interval. All these libraries use WebSocket protocol for the data transfer.

In the solution based on mutation-summary [48] JavaScript library, faye-websocket library is used as the message broker. We created two web applications out of which

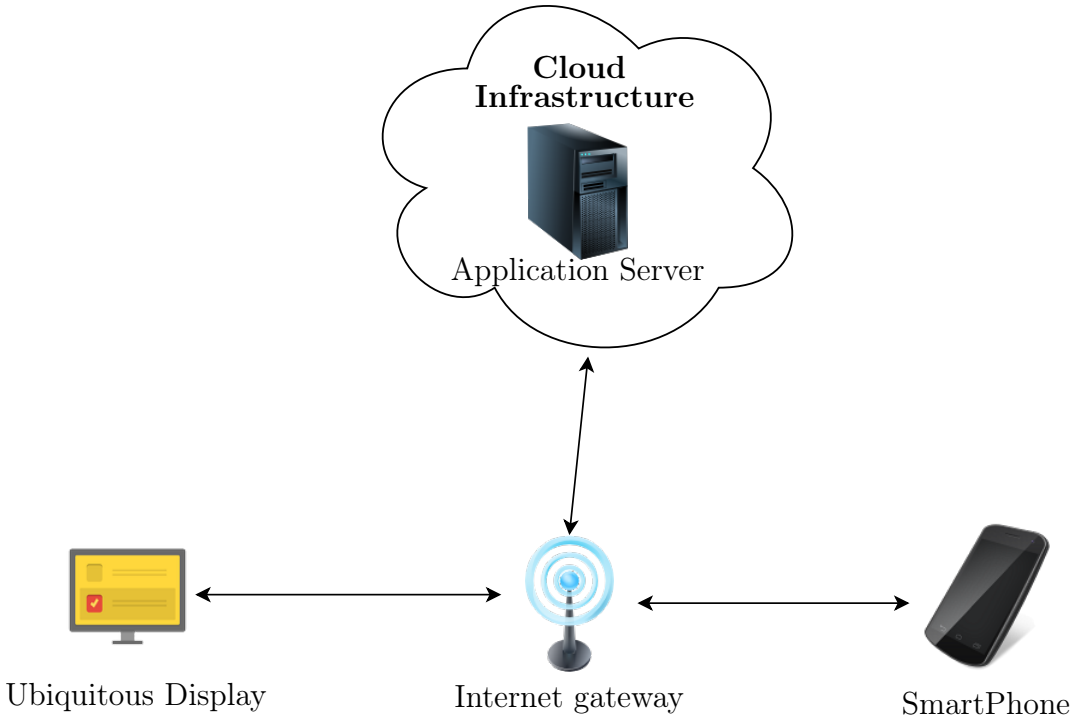



Figure 9: System Architecture

one application contains the actual content and the other application is just a mirror page that shows the cloned content. Both these applications are deployed to the cloud infrastructure. The mobile client accesses the application with the actual content. The other client laptop accesses the cloned content. Messages are exchanged in publish/subscribe topic based pattern over Faye's WebSocket implementation.

In the solution based on ShareJS [23] JavaScript library, livedb is used as the message broker that provides publish/subscribe mechanism. We used 'ws' a WebSocket implementation library to use the WebSocket functionality.

In the solution based on Socket.io [56], engine.io is used for the WebSocket functionality for the transport.

For all the implementations mobile device is the publisher (i.e., client that publishes messages to the channel) and ubiquitous display is the subscriber (i.e., client that receives the messages through the subscribed channel) as shown in Figure 10.

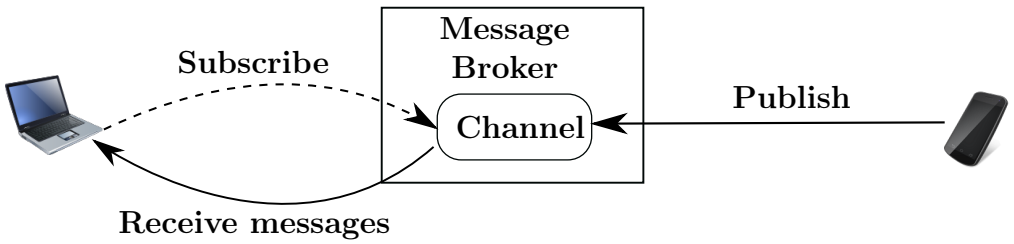



Figure 10: Publish/Subscribe Architecture common to all the implementations

The performance of the application for each of these libraries is evaluated to find

the best suitable library for the ubiquitous display networks in terms of overhead, latency and page load time.

Further we used VNC with WebSocket to study and compare the performance of the application and the plot obtained is common to all the implementations.

### 4.2.1 VNC with WebSocket

In this experiment the payload is measured when the data is sent from publisher to subscriber using VNC with WebSocket. In this experiment VNC server is run on client 2 (Laptop). The VNC server used is 'X11VNC' and the version is 0.9.13. A proxy server is installed on Laptop that accepts websocket connections. Also, a HTML5-based VNC client application called 'noVNC' is installed on the Laptop. Mobile client uses the VNC client application from a web browser and after successful authentication accesses the content of the Laptop. Password based authentication is used and the connection is secured over TLS. The experiment setup is as shown in Figure 11.

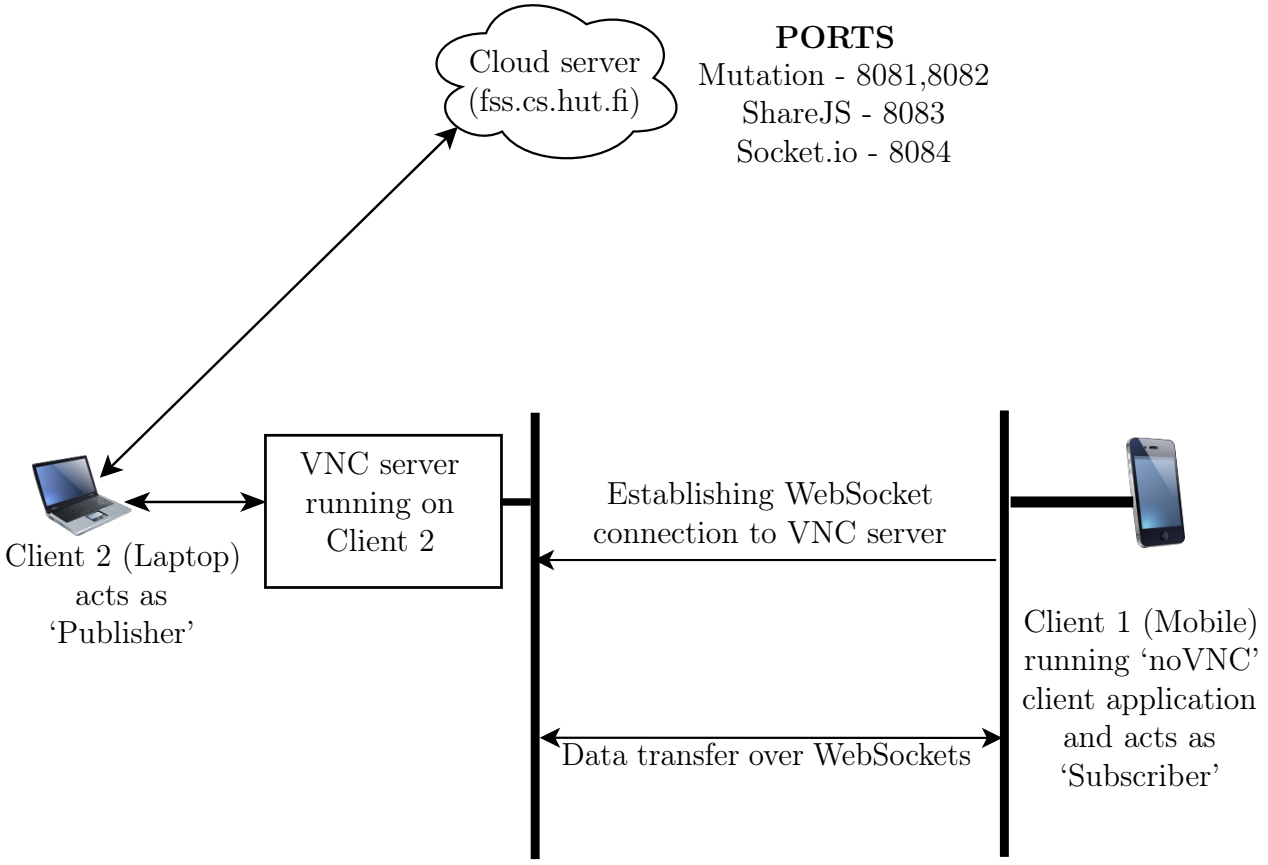



Figure 11: VNC with WebSocket

## 4.3 Experimental Setup

The application and other equipment used for the experiment is as follows:

- The HTML5 presentation application implemented using three libraries mutation-summary, ShareJS and Socket.io as mentioned earlier in this section. The time intervals chosen for the automatic state change logic for the web application are 1s, 5s, 10s, 20s, 30s, 40s, 50s, 60s. For each of these intervals, slide changes in the web application. The web application is accessed by both the laptop and the mobile device. For a particular interval set in the web application, the state change event is triggered from the mobile. This setup is kept for 1 hour for each interval to capture the packets.

- Cloud Infrastructure (VirtualMachine) is used to deploy the web application. The VM instance is created in CSC cPouta service. The OS running on the server is Ubuntu and the version is 14.04.2 LTS. For the application implemented using the mutation-summary library, two ports are reserved: port 8081 is used by the application that displays the cloned content, and port 8082 is used by the application that mirrors the content. We used two ports for the mutation summary implementation just to give an example on how the content cloning works. The actual content contains HTML5 presentation application and this content is cloned to the mirrored page. The mirrored page also contains the cloned slide changes. The application implemented using ShareJS library is run on port 8083. The application implemented using Socket.io is run on port 8084.

- The browser used for the experiment is Chrome and Version 44.0.2403.155 (64-bit).

- The mobile used for the experiment is Nexus 5 and the OS installed on it is Android 5.1.1. This device is called Client 1 in the rest of the thesis.

- The laptop used is Dell Inspiron series and model number is N5010. The OS installed on it is Ubuntu 14.04 LTS. This device is called Client 2 in the rest of the thesis.

We considered Wi-Fi interface for our experiment and the specifications of the interface are mentioned in the below table:

| Interface | Wi-Fi |
|---|---|
| Download speed | 30.29 Megabits per second |
| Upload speed | 39.99 Megabits per second |
| Radio Type | 802.11n |
| SSID | aalto open |
| Authentication type | open |

Table 1: Wi-Fi Interface Specifications

The packets are captured at the cloud server when Wi-Fi interface is used.

### 4.3.1 Metrics

To analyze the performance of the application in pervasive display networks and also to benchmark the JavaScript libraries used in the implementations, we focused on evaluating the following metrics:

- Page load time of the web application. We used a library called phatomas [35] to get a report on the page loading time statistics. The experiment is done by initiating 10 requests for the three versions of the solution. The generation of multiple requests is automated using the script provided in the open source implementation. The final plot gives visual representation of various metrics related to web application performance.

- WebSocket frames payload is analyzed to get an idea on the actual WebSocket payload data that is being carried over the wire. WebSocket Frames payload is plotted by extracting the WebSocket protocol related information from the captured pcap file using tshark. The resulting csv file is filtered per hostname (fss.cs.hut.fi) and port (8081/8082/8083/8084). The filtered data is then plotted.

- Application response time is the time taken by an event triggered by client 'a' to reflect on the client 'b'. This metric is studied to understand how fast a user can see the application state changes that are sent by another user, when both of them are connected using WebSocket for real-time communication. To measure this metric an npm library, timesync is used. A timesync client connects to the server and will synchronize its time. When a publisher (mobile) sends events, the timestamp of the timesync server at that particular moment is taken, and sent to the subscribers along with the message. On receiving the message from the publisher, the subscriber (Laptop) then calculates the application response time by taking the difference between the timestamp of the timesync server, at that moment when it received the message and the timestamp that it received in the message sent by the other client. This delay is represented as the application response time in applying the changes to the pervasive displays.

- Maximum, minimum, and average values of round trip times (RTT) per connection. These round-trip-time (RTT) measurements are useful to study the delay between the client and server communication.

- The average throughput is calculated as the unique bytes sent divided by the elapsed time i.e., the value reported in the unique bytes sent field divided by the elapsed time (the time difference between the capture of the first and last packets in the direction) is used for this measurement.

- TCP payload data, which are unique bytes sent (i.e., total bytes sent excluding retransmitted bytes and bytes due to window probing). TCP data payload metric gives the actual data being transmitted without the overhead and can be used to analyze the bandwidth requirements in display networks.

- Actual data packets (i.e., packets with atleast a byte of TCP data payload are considered).

Tcptrace [46] utility is used to generate csv files from the captured pcap files. These csv files are then filtered per hostname (fss.cs.hut.fi) and port (8081, 8082/8083/8084). After pre-processing the data, the resulting TCP metrics are plotted for further analysis.

## 4.4 Results

All the following plots are plotted with the data captured over the Wi-Fi interface.

### 4.4.1 Page Loading Time

The plot in Figure 12 shows the distribution of page loading time. We measured "DOM complete" metric that gives the time taken by the entire HTML document and all sub-resources to finish loading.

From Figure 12 we infer that the maximum, minimum, average and median values of DOM complete loading time are high in the mutation-summary based implementation. The reason is that in the mutation-summary based solution entire DOM content is cloned including the CSS styling and takes more time to load the page. The Socketio based solution has the least maximum loading time value compared to the other solutions. However, shareJS has least average, minimum and median loading times. Hence from the above comparisons ShareJS has the least page loading time because of the reason that sharejs uses Operational Transformation algorithms to synchronize the DOM content.

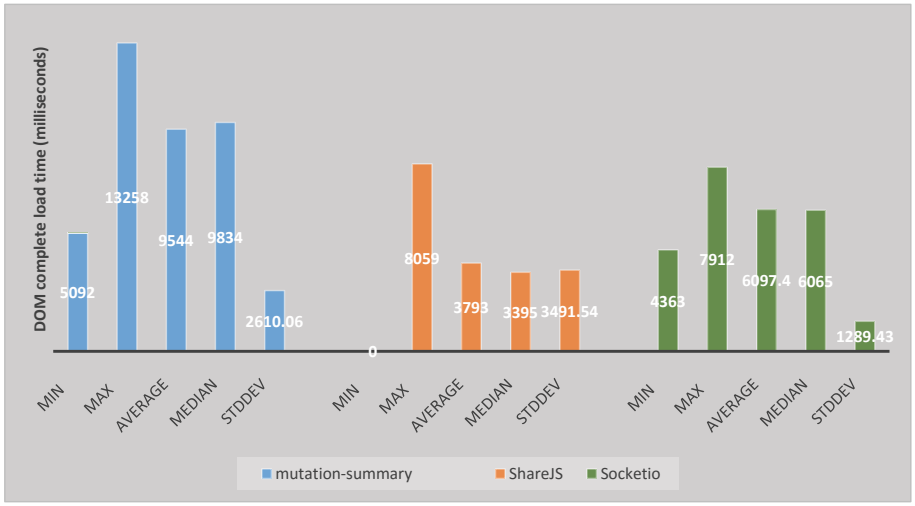



Figure 12: DOM complete load time for all the implementations

### 4.4.2 WebSocket Frames Payload

We considered WebSocket frames payload for the analysis. We analyze five parameters also called five point summary, which are smallest value, largest value, first quartile (i.e., value of 25 percentage of the observations), median (i.e., value of 50 percentage of the observations), third quartile (i.e., value of 75 percentage of the observations), inter-quartile (i.e., difference between third and first quartile values), and skewness of the distributed data i.e., skewed right (observations concentrated on lower end of the scale) or skewed left (observations concentrated on higher end of the scale) or symmetric (observations spread equally at the median).

The plot of WebSocket Frames payload metric for the mutation-summary based solution is shown in Figure 13. The distribution of data is skewed left except at 60s interval that has symmetric distribution. The maximum value of the data is almost same at all the event intervals which is about 125 bytes.

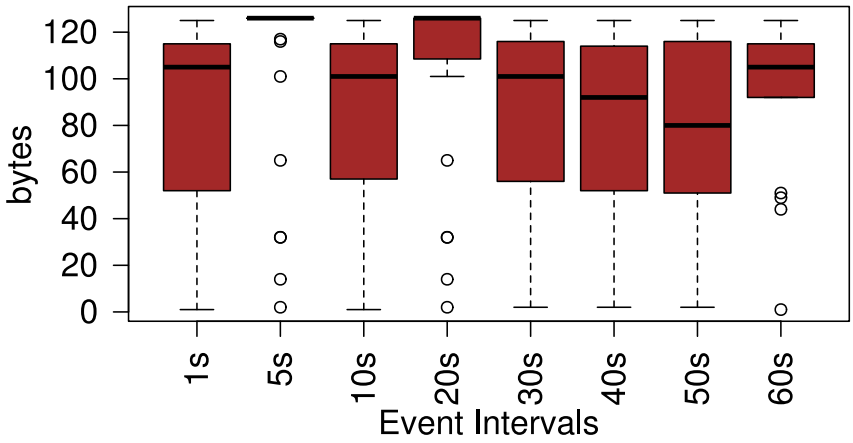



Figure 13: WebSocket Frames Payload for the solution based on mutation-summary library

Following are the conclusions made from the plot for the mutation-summary based solution:

- The maximum, minimum and inter quartile values are almost same at all the intervals considering the outliers. From this we can say that the number of bytes transferred is almost consistent at all the intervals. This is mainly because when there is a slide change, the corresponding HTML DOM element properties change and this is sent as a message over WebSocket connection.
- The maximum payload transfer is about 130 bytes in this solution.
- The median value has fluctuated until 30s event interval, then it has a downward trend till 50s and at 60s interval the value has increased. From these observations

we conclude that there is not much variation in the median value for most of the intervals.

The plot of WebSocket Frames payload metric for the ShareJS based solution is shown in Figure 14. From the box-plot we observe that the payload median value is about same i.e., 20 bytes at all the intervals except at 5s that has a value of 120 bytes. The distribution of data is skewed right at all the intervals except at 5s that has skewed left distribution. The inter-quartile range value is approximately 105 bytes at all the intervals. Also, there are no outliers in this plot.

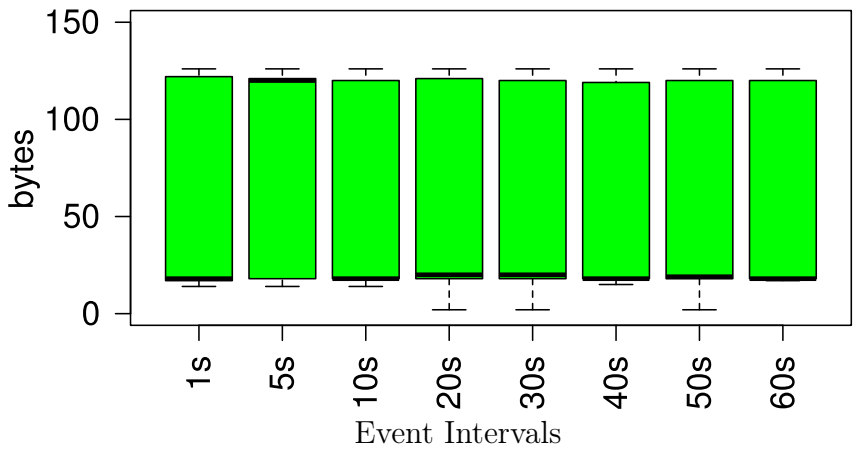


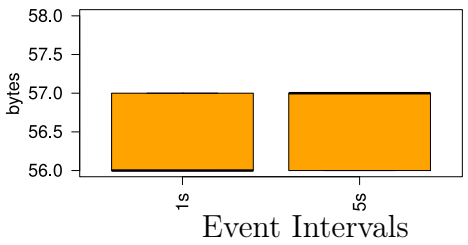

Figure 14: WebSocket Frames Payload for the solution based on ShareJS library

Following are the conclusions for the ShareJS based solution from the plots:

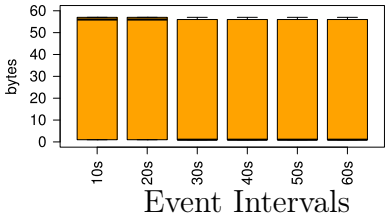
- The maximum, minimum and inter quartile values are almost same at all the intervals. This indicates that the distribution of data is consistent at all the event intervals except at 5s interval. At 5s interval, the distribution is skewed left because a particular slide change that contains largest resource has occurred more frequently.
- The maximum payload transfer is about 130 bytes in this solution. There is not much variation in the five point summary characteristics except at 5s interval.

The plot of WebSocket Frames payload metric for the Socketio based solution is shown in Figures 15a and 15b. Figure 15a shows the box-plot data at 1s and 5s intervals. Figure 15b shows the box-plot data for rest of the intervals. From the box-plot we observe that the payload median value has a large variation across all the event intervals and its value coincided with either maximum or minimum value of the distributed data. The payload median value at 5s, 10s and 20s intervals is about 57 bytes and at 1s interval it is about 56 bytes. The data is skewed left at 5s,

10s, and 20s intervals and skewed right for rest of the intervals. The inter-quartile range value is approximately between 1 byte for 1s and 5s intervals and has a value of 57 bytes approximately for the other intervals.




(a) For the intervals 1s and 5s




(b) For the intervals 10s, 20s, 30, 40s, 50s, 60s

Figure 15: WebSocket Frames Payload for the solution based on Socketio library

From the plot we infer that for the Socket.io based solution the maximum payload transfer is about 60 bytes.

The plot of WebSocket Frames payload metric for the VNC with WebSocket based solution is shown in Figure 16. From the box-plot we observe that the payload median value is almost same i.e., about 55 bytes at all the event intervals. The distribution of data is slightly skewed right at all the intervals. The inter-quartile range value is between 60-70 bytes approximately at all the intervals.

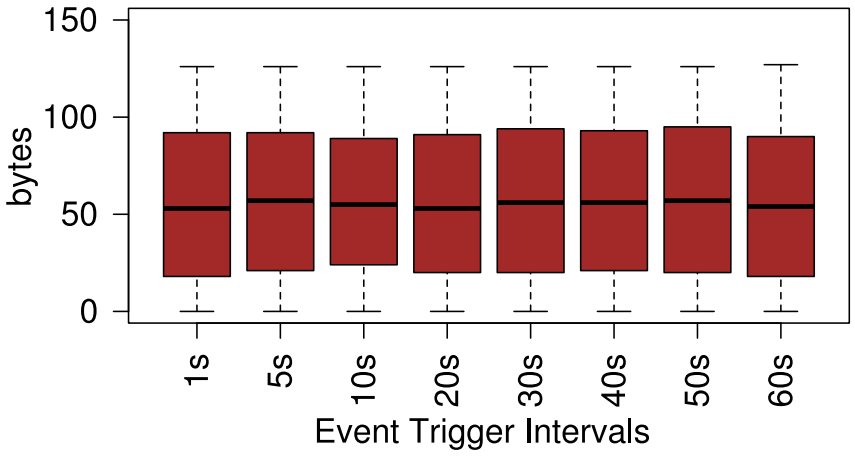



Figure 16: WebSocket Frames Payload for the VNC solution

There is not much variation in VNC with WebSocket based solution as the five point summary values of this metric is almost same at all the event intervals.

On the whole, our inference from the WebSockets frames payload metric related plots is as follow:

- VNC based solution has maximum payload transfer when compared to other solutions.
- Mutation-summary based solution has higher values when compared to ShareJS and Socket.io based solutions. This can potentially be explained since the mutation-summary library changes the DOM sub-tree for a state change. This leads to a large payload transfer over the wire. However in ShareJS and Socketio only the state change message is sent over the wire and the receiving client renders the DOM based on the received message.
- Socketio based solution has the least payload transfer value compared to other solutions.

### 4.4.3 Application Response Time

The plot of application response time metric for the mutation-summary based solution is shown in Figure 17. From the box-plot median value of the metric is between 10-38 ms approximately at all the event intervals. The distribution of data is skewed right at all the intervals except at 40s and 50s that has symmetric distribution of data. The inter-quartile range value is between 9-20 ms approximately at all the intervals.

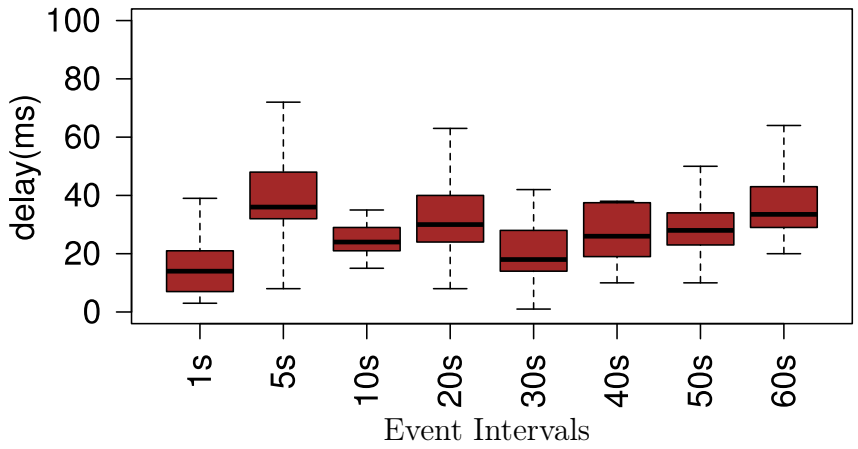



Figure 17: Application response time for the solution based on mutation-summary library

The plot of application response time metric for the ShareJS based solution, is shown in Figure 18. From the box-plot median value of the metric is between 200-1100ms at all the event intervals. The distribution of data is skewed right at all

the intervals. The inter-quartile range value is between 150-1700 ms approximately at all the intervals.

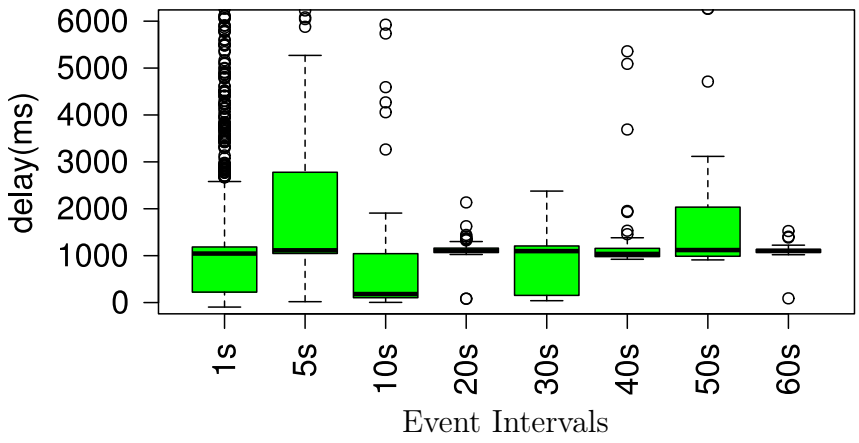
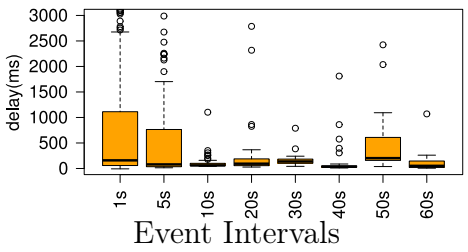
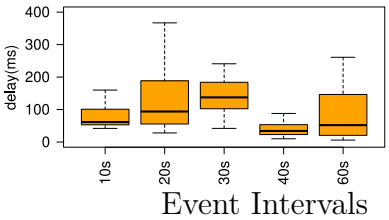



Figure 18: Application response time for the solution based on ShareJS library

The plot of application response time metric for the Socket.io based solution, is shown in Figures 19a and 19b. From the box-plot median value of the metric is between 100-300 ms at 1s, 5s, and 50s event intervals and at 10s, 20s, 30s, 40s, and 60s intervals the value is between 30-130 ms approximately. The distribution of data is skewed right at all the intervals except for the 30s that have symmetric distribution. The inter-quartile range value is 330-1100ms at 1s, 5s, and 50s event intervals and at 10s, 20s, 30s, 40s, and 60s intervals the value is between 50-120 ms approximately.




(a) For all the intervals respectively




(b) For intervals 10s, 20s, 30s, 40s, 60s respectively

Figure 19: Application response time for the solution based on Socketio library

On the whole, our inference from the application response time metric related plots is as follows:

- There is a fluctuation in the median value at all event intervals, in all the three versions of the solution.
- Both the ShareJS and Socket.io based solutions have large application response times that can reach upto 3000ms in worst case scenario.
- Mutation-summary based solution has less delay compared to the other two solutions. This is because of the reason that in this solution only the net changes are sent over the wire. This algorithm is faster in synchronizing changes compared to the other solutions.
- The delay value is large in ShareJS and Socketio based solutions because of the reason that the message brokers are different in these solutions.
- Outliers observed in ShareJS and Socketio based solutions are more in number at 1s event intervals. In both the solutions these outliers fluctuate over the event intervals. The reason for more number of outliers is that at 1s event interval the state changes happen quickly and if there is network delay then the message transfer delay increases as the messages are queued. This is applicable to highly interactive applications and the response will be slower in those applications.

### 4.4.4 Round Trip Time (RTT)

The plot of Round-Trip Time (RTT) average value for client 1 and for all the solutions, is shown in Figure 20. From the box-plot the approximate median values of the metric data for the mutation-summary, ShareJS and Socket.io based solutions over the intervals are in the range 0.5-1.9 ms, 0-0.8 ms, and 0-0.5 ms respectively. The approximate inter-quartile values in mutation-summary, ShareJS and Socket.io based solutions over the intervals are in the range 1.6-3.2 ms, 0.2-2.6 ms, 0.2-0.8 ms respectively. The distribution of data for the solutions is skewed right at all the intervals except at 5s and 10s that has a skewed left distribution for ShareJS based solution.

The plot of Round-Trip Time (RTT) average value for client 2 and for all the solutions, is shown in Figure 21. From the box-plot the approximate median values of the metric data for the mutation-summary, ShareJS and Socket.io based solutions over the intervals are in the range 0.8-1.8 ms, 0-1.8 ms, and 0-1.3 ms respectively. The approximate inter-quartile values in mutation-summary, ShareJS and Socket.io based solutions over the intervals are in the range 1-2 ms, 1-2.8 ms, 1-3 ms respectively. The distribution of data for the solutions is skewed right at all the intervals except at 1s and 20s. At the 1s interval mutation summary based solution has a skewed right distribution and in the 20s interval ShareJS had symmetric distribution.

Inference from the RTT average metric related plot is as follows:

- Mutation-summary based solution has slightly higher RTT average value at all the event intervals compared to other solutions. This could be because of

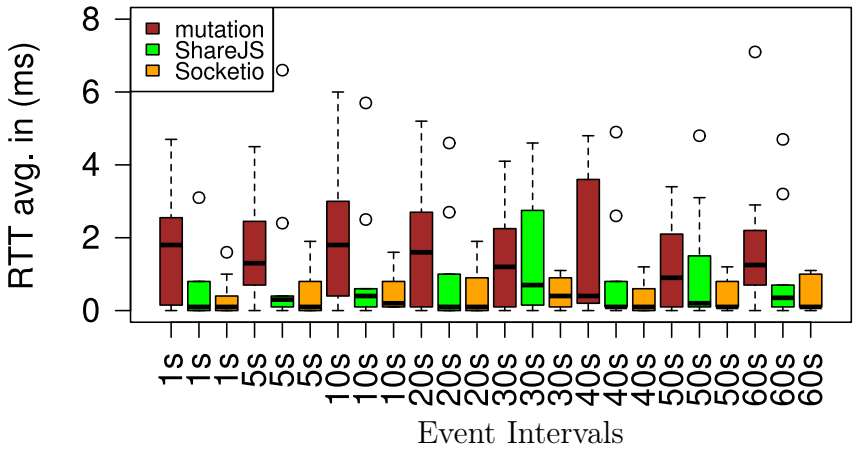



Figure 20: RTT average for client 1

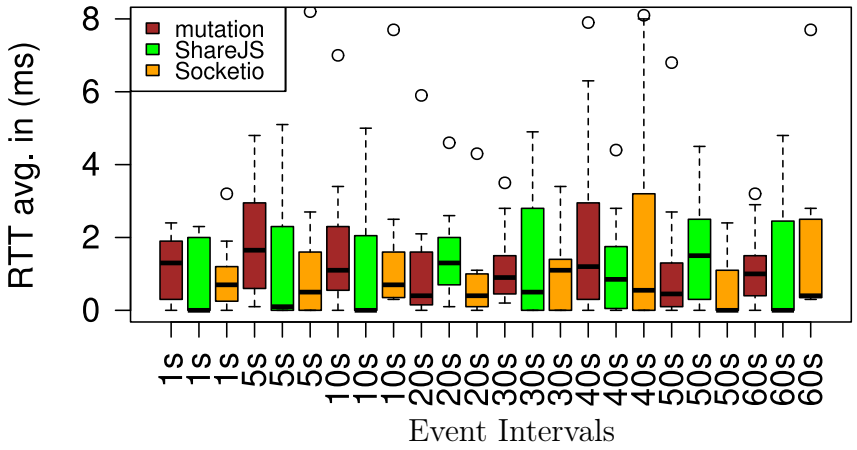



Figure 21: RTT average for client 2

the reason that the data transfer is high in this version of the solution and is inferred from data related to WebSocket frames payload metric. Also, the RTT average value is less than 6 milliseconds in all versions of the solution.

- Fluctuations are observed in the median values for the mutation-summary based solution. For the other solutions the value is almost skewed to the lower value which is approximately 1 milliseconds for both the clients.

- For client 2 the maximum value of RTT average is observed at 40s event interval

for the Socketio based solution. This spike in the maximum value could be due to network disruption and the client took more time to receive a response from the server.

- The outliers fluctuate over the intervals for both the clients.

The plot of Round-Trip Time (RTT) maximum value for client 1 and for all the solutions, is shown in Figure 22. From the box-plot the approximate median values of the metric data for the mutation-summary, ShareJS and Socket.io based solutions over the intervals are in the range 2-6 ms, 0-2 ms, and 0-1 ms respectively. The approximate inter-quartile values in mutation-summary, ShareJS and Socket.io based solutions over the intervals are in the range 4-9 ms, 2-6 ms, 1-3 ms respectively. The distribution of data for the solutions is skewed right at all the intervals.

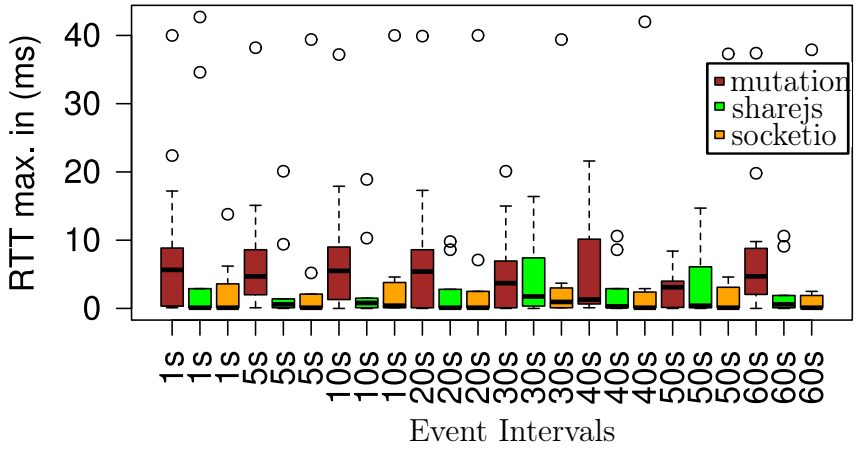



Figure 22: RTT maximum for client 1

The plot of Round-Trip Time (RTT) maximum value for client 2 and at all the solutions, is shown in Figure 23. From the box-plot the approximate median values of the metric data for the mutation-summary, ShareJS and Socket.io based solutions over the intervals are in the range 2-7 ms, 0-5 ms, and 2-5 ms respectively. The approximate inter-quartile values in mutation-summary, ShareJS and Socket.io based solutions over the intervals are in the range 1-10 ms, 3-13 ms, 4-22 ms respectively. The distribution of data for the solutions is skewed right at all the intervals. The inter-quartile range value for the solutions over the intervals is between 3-23 ms approximately.

Inference from the RTT maximum metric related plot is as follows:

- For client 1, mutation-summary based solution has higher five point summary values. For client 2 these values are higher for the Socket.io based solution. On

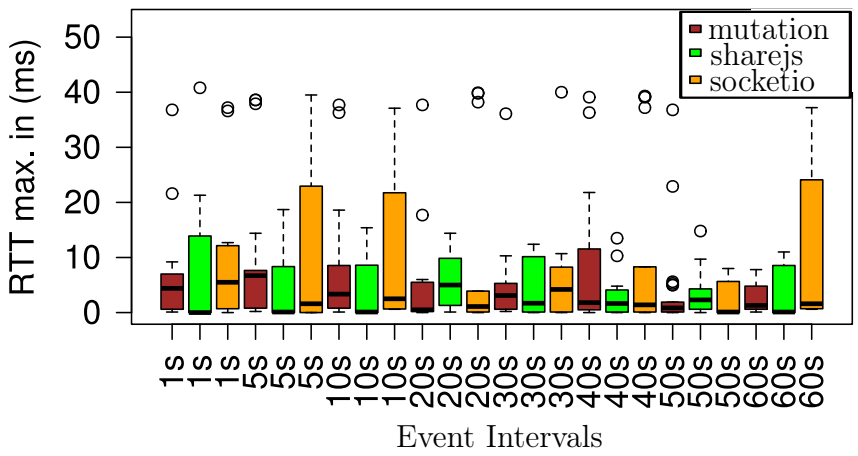



Figure 23: RTT maximum for client 2

the whole mutation-summary based solution has slightly higher RTT maximum values for all the event intervals compared to the other solutions.

- For client 1, mutation-summary based solution has a median value of around 5 milliseconds for all the event intervals except at 40s and 50s, where the values are below 5 milliseconds. For the other versions there is not much variation in the median value and is less then 4 milliseconds.

- For client 2, the median value variation is similar to that of the client value and the value is less than 8 milliseconds for all the solutions. Also, we noticed that Socketio based solution has much skewness and could be due to delay in fetching the large page resources.

- For the outliers maximum value is about 40 milliseconds at all the event intervals, for both the clients and for the versions based on mutation-summary and Socketio. The outliers in Sharejs has less maximum value compared to the other two versions of the solution.

The plot of Round-Trip Time (RTT) minimum value for client 1 and for all the solutions, is shown in Figure 24. From the box-plot we observe that for all the solutions the median, maximum, minimum, and quartile values is between 0-0.1 ms. Overall RTT minimum value observed is around 0 ms for all the solutions.

The plot of Round-Trip Time (RTT) minimum value for client 2 and for all the solutions, is shown in Figure 25. From the box-plot we observe that for all the solutions the median, maximum, minimum, and quartile values is between 0-0.6 ms. Overall RTT minimum value observed is around 0 ms for all the solutions.

For client 1 the ShareJS and Socket.io based solutions have higher values. For client 2 the values are higher for the mutation-summary based solution. On the

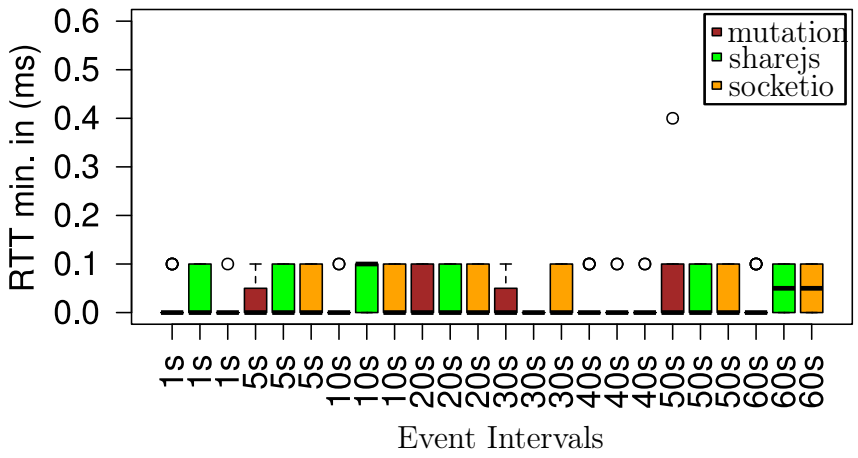



Figure 24: RTT minimum for client 1

whole mutation-summary based solution has slightly higher values compared to the other solutions.

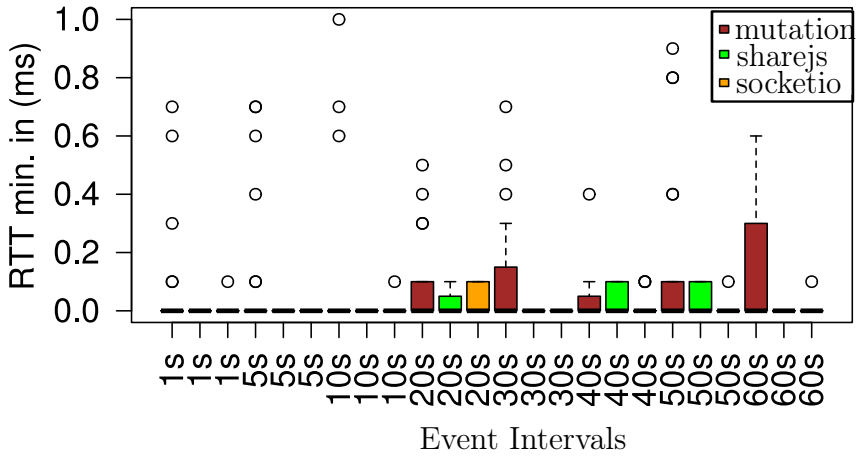



Figure 25: RTT minimum for client 2

From the RTT minimum plots we observe that the minimum RTT seen in the RTT samples is less than 0.1 milliseconds for both the clients and at all the event intervals. The RTT minimum value seen is minimal due to the location of the cloud server instance and also depending on the type of implementation. The outliers are observed more for client 2 compared to client1.

From all the RTT measurements we observe that the mutation-summary based

solution has higher RTT values compared to other versions of the solution. The reason could be that the websocket implementation in this library is different from the other libraries and uses faye-websocket library for the communication. The other reason could be that the library has many resources to load before sending response to the client. This pattern is also observed in the page load metric discussed earlier. However, lower RTT values are observed in this implementation and can still be used for real-time applications. The other libraries ShareJS and Socket.io, though use different websocket implementations, perform faster and contain less resources to load before sending response to the client.

### 4.4.5 Unique data bytes in TCP payload

The plot of unique data bytes value for the mutation-summary based solution, is shown in Figure 26. From the box-plot median value of the metric is between 600-700 bytes at all the event intervals. The distribution of data is skewed right at all the event intervals except at 1s and 5s interval where the distribution is symmetric and skewed left respectively. The inter-quartile range value is between 50-500 bytes approximately at all the intervals.

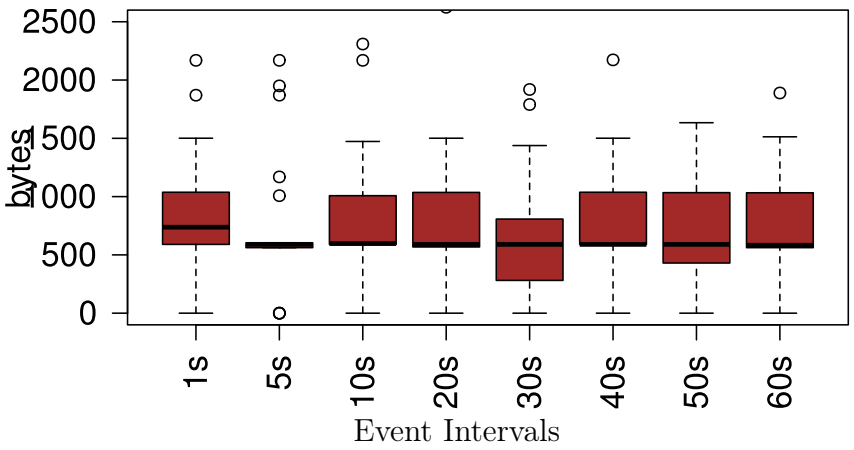



Figure 26: Unique Bytes in TCP payload for the mutation-summary based solution

The plot of unique data bytes value for the ShareJS based solution, is shown in Figure 27. From the box-plot median value of the metric is between 100-1000 bytes at all the event intervals. The distribution of data is skewed right at all the event intervals except for the 10s and 40s interval where the distribution is symmetric. The inter-quartile range value is between 1000-7000 bytes approximately at all the intervals.

The plot of unique data bytes value for the Socket.io based solution, is shown in Figure 28. From the box-plot median value of the metric is consistent, that is

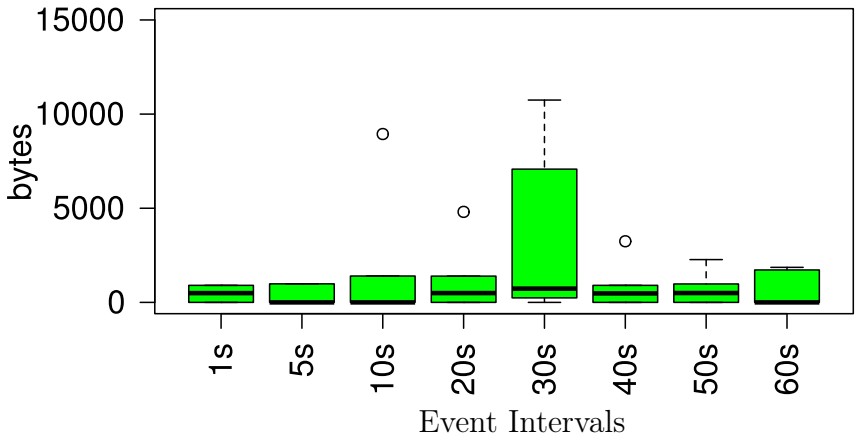



Figure 27: Unique Bytes in TCP payload for the ShareJS based solution

about 500 bytes at all the event intervals except at 10s interval that has a value of 0 bytes. The distribution of data is skewed right at all the event intervals except for the 60s where the distribution is symmetric. The inter-quartile range value is between 1000-2000 bytes approximately at all the intervals.

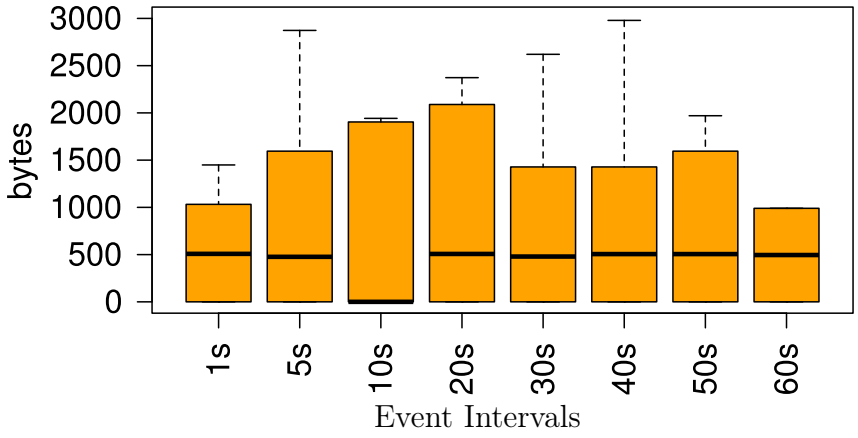



Figure 28: Unique Bytes in TCP payload for the Socketio based solution

On the whole, our inference from the unique bytes related plots is as follows:

- The unique bytes values are higher in the Socket.io based solution compared to other solutions.

- We could notice that in the ShareJS based solution the five point summary values for this metric are higher only at a particular random event interval i.e., at 30s. It could be due to increase in the number of operations generated for the corresponding state change.
- There is not much variation in the median value for all the three versions i.e., around 500 bytes.
- Socketio based solution observed fluctuations in the maximum value.
- The outliers are observed more in mutation-summary based solution compared to the other solutions. There are slight fluctuations in the outliers in this solution.

### 4.4.6 Actual data packets in TCP payload

The plot of actual data packets value for the mutation-summary based solution, is shown in Figure 29. From the box-plot median value of the metric is consistent, that is around 2 packets at all the event intervals. The distribution of data is skewed right at all the event intervals except at the 30s and 50s intervals where the distribution is symmetric. The inter-quartile range value is between 1-2 packets approximately at all the intervals.

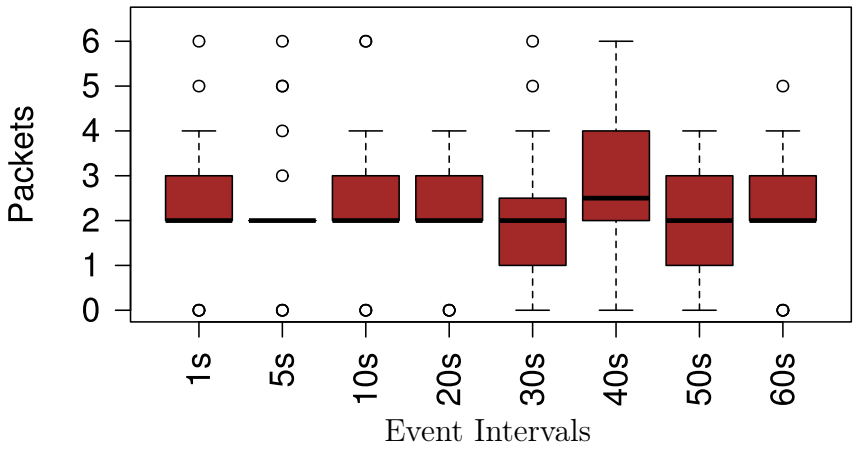
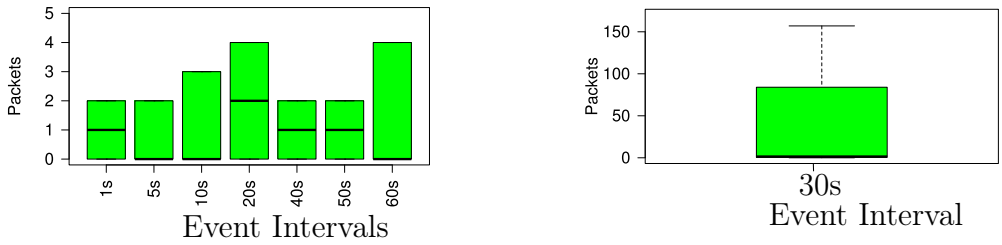



Figure 29: Actual Packets in TCP payload for the mutation-summary based solution

The plots of actual data packets value for the ShareJS based solution, is shown in Figures 30a and 30b. From the box-plot median value of the metric is between 0-2 packets at all the event intervals. The maximum, minimum, third quartile and first quartile values for the solution at 30s interval are 150, 0, 80, 0 packets respectively. The inter-quartile value is between 2-4 packets at all the intervals except at 30s

interval where the value is about 70 packets. The distribution of data is symmetric at 1s, 20s, 40s, 50s intervals and at 5s, 10s, 30s and 60s intervals it is skewed right.




(a) For intervals 1s, 5s, 10s, 20s, 40s, 50s, 60s (b) For 30s interval

Figure 30: Actual Packets in TCP payload for the ShareJS based solution

The plot of actual data packets value for the Socket.io based solution, is shown in Figure 31. From the box-plot median value of the metric is having a constant value of 1 packet for all the event intervals except at the 10s interval where the value is 0 packet. The distribution of data is skewed right at all the event intervals except for 60s where it is symmetric. The inter-quartile range value is between 2-4 packets approximately at all the intervals.

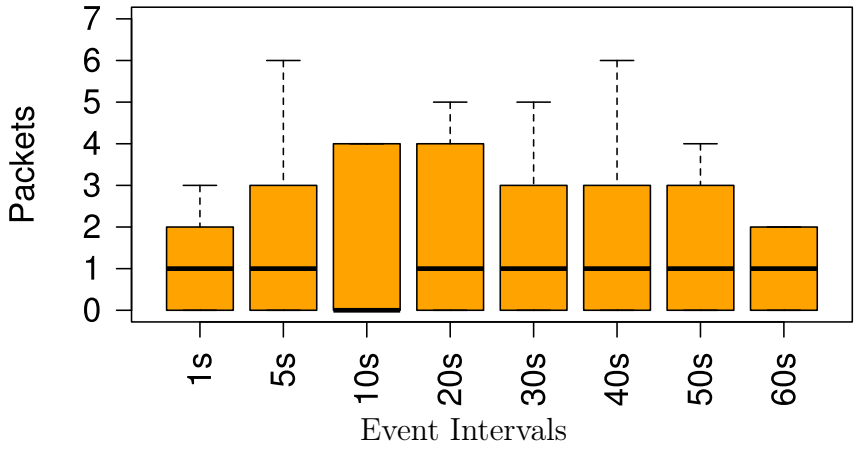



Figure 31: Actual Packets in TCP payload for the Socketio based solution

On the whole, our inference from the actual data packets metric related plots is as follows:

- The median values are almost constant at all the event intervals for Socketio and mutation-summary based solutions.

- For the shareJS based solution fluctuations are observed for the five point summary values of this metric.
- The five point summary values had a sudden peak at a particular event interval in the ShareJS based solution i.e., at 30 second. This is also the reason for the high unique bytes value at 30s interval that is mentioned earlier. The reason could be due to increase in the number of operations for the corresponding state change. For rest of the event intervals these values are low.
- The number of packets transferred is high in the mutation-summary based solution among other solutions considering the median values of the metric.
- The outliers are more in mutation-summary based solution compared to the other solutions. However, the number of outliers are minimal. Considering these outliers we can say that at any given event interval a maximum of 6 packets are transferred.

### 4.4.7 Throughput

Throughput plots are considered without outliers as we observed minimal outliers and are almost negligible. The outliers are excluded using R boxplots for the reason that we have sufficient data to analyze and to reduce the skewness of the data. The plot of throughput value for client 1 and for the mutation-summary based solution, is shown in Figure 32. From the box-plot median value of the metric is between 0-10000 bytes/second at all the event intervals. The distribution of data is skewed right at all the event intervals except at 60s interval where the distribution is skewed left. The inter-quartile range value is between 2000-12500 bytes/second approximately at all the intervals.

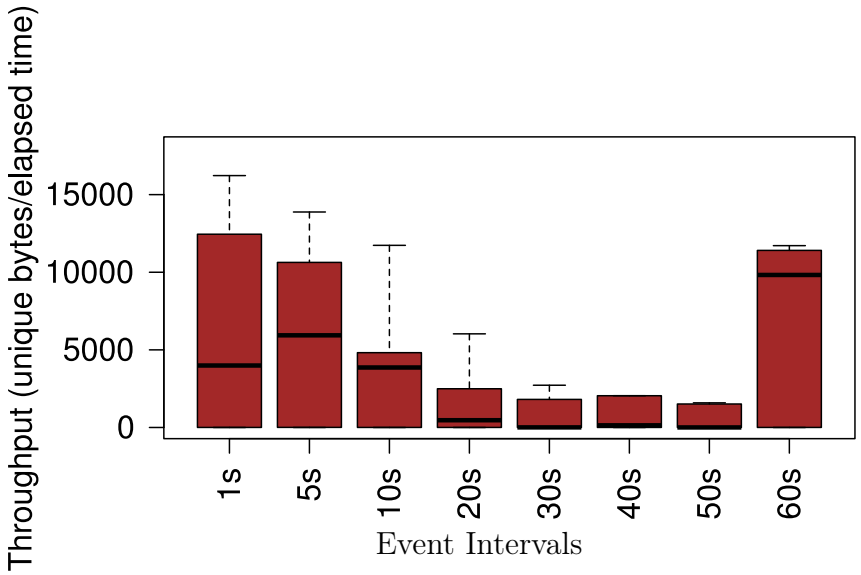



Figure 32: Throughput for client1 and for the mutation-summary based solution

The plot of throughput value for client 2 and for the mutation-summary based solution, is shown in Figure 33. From the box-plot, median value of the metric is between 0-14000 bytes/second at all the event intervals. The distribution of data is skewed right at all the event intervals except at 50s and 60s intervals where the distribution is skewed left. The inter-quartile range value is between 2500-14000 bytes/second approximately at all the intervals.

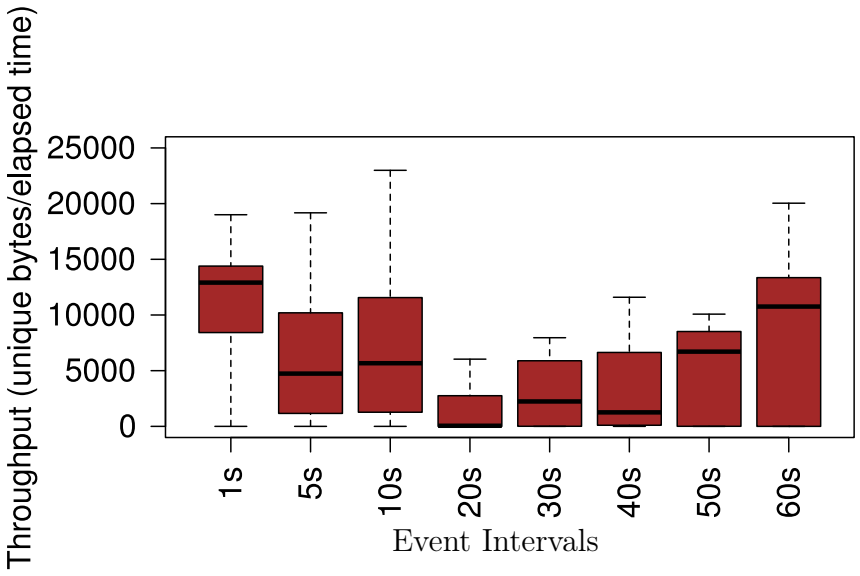



Figure 33: Throughput for client2 and for the mutation-summary based solution

The plot of throughput value for client 1 and for the ShareJS based solution, is shown in Figure 34. From the box-plot, median value of the metric is between 0-4 bytes/second at all the event intervals. The distribution of data is skewed right at all the event intervals except at 20s and 50s interval where the distribution is skewed left. The inter-quartile range value is between 4-10 bytes/second approximately at all the intervals.

The plot of throughput value for client 2 and for the ShareJS based solution, is shown in Figure 35. From the box-plot, median value of the metric is between 3-9 bytes/second at all the event intervals. The distribution of data is skewed right at all the event intervals. The inter-quartile range value is between 2-10 bytes/second approximately at all the intervals.

The plot of throughput value for client 1 and for the Socket.io based solution, is shown in Figure 36. From the box-plot, median value of the metric is between 0-5 bytes/second at all the event intervals. The distribution of data is skewed right at all the event intervals except at 20s and 40s intervals where the distribution is skewed left. The inter-quartile range value is between 4-11 bytes/second approximately at all the intervals.

The plot of throughput value for client 2 and for the Socket.io based solution, is shown in Figure 37. From the box-plot, median value of the metric is between 4-10 bytes/second at all the event intervals. The distribution of data is skewed right at all the event intervals except at 30s interval where the distribution is skewed left.

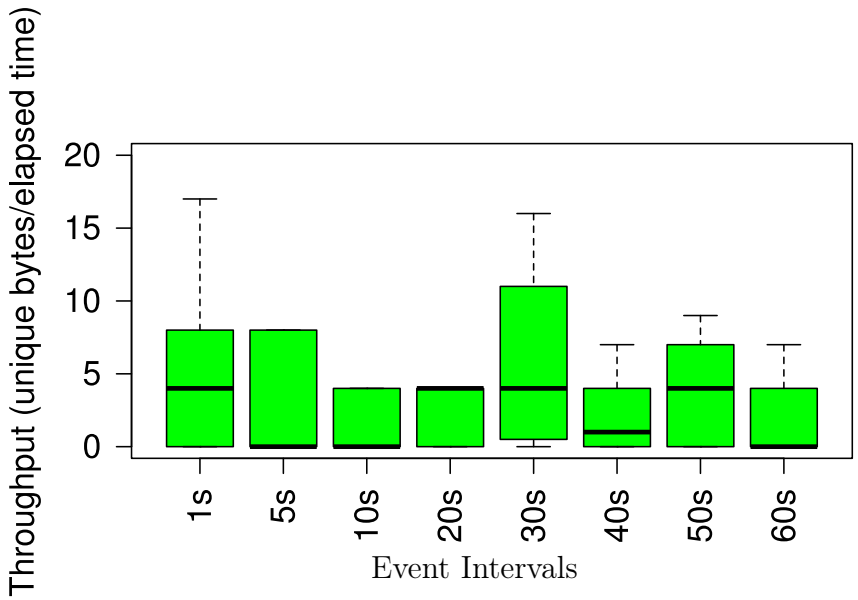



Figure 34: Throughput for client1 and for the ShareJS based solution

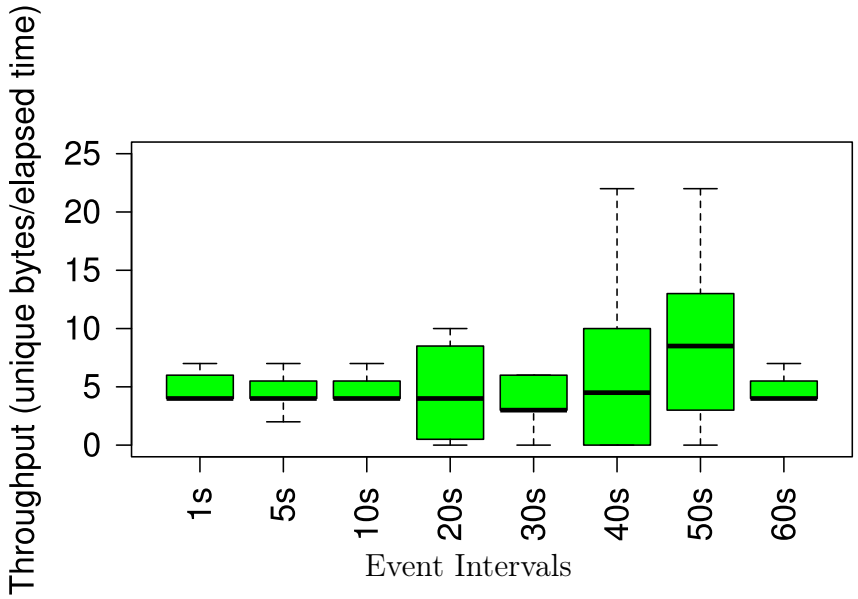



Figure 35: Throughput for client2 and for the ShareJS based solution

The inter-quartile range value is between 5-40 bytes/second approximately at all the intervals.

On the whole, our inference from throughput related plots is as follows:

- Mutation-summary based solution has higher throughput compared to other solutions. This could be because of the amount of data that is transferred for a corresponding state change. For client 1, a decreasing trend is observed for this metric value until 50s interval and then there is a sudden increase in the

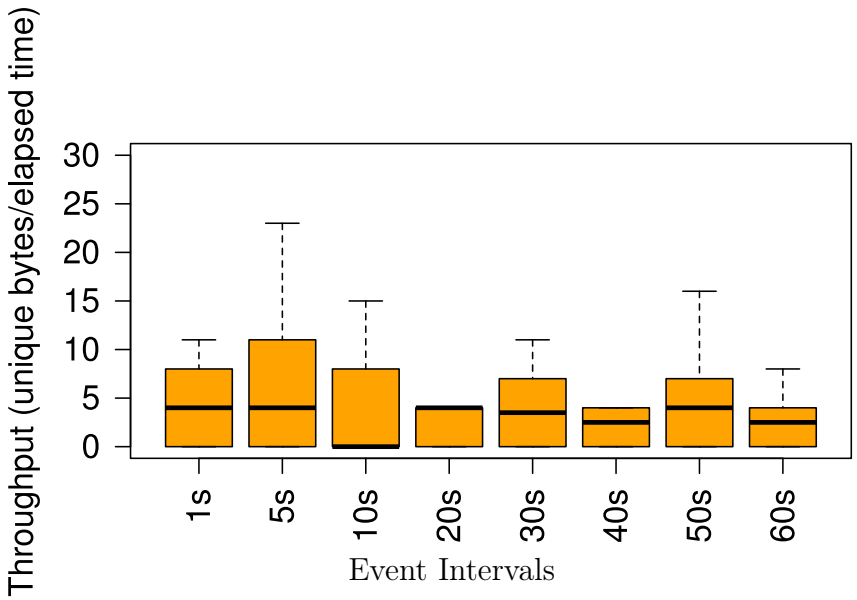



Figure 36: Throughput for client1 and for the Socketio based solution

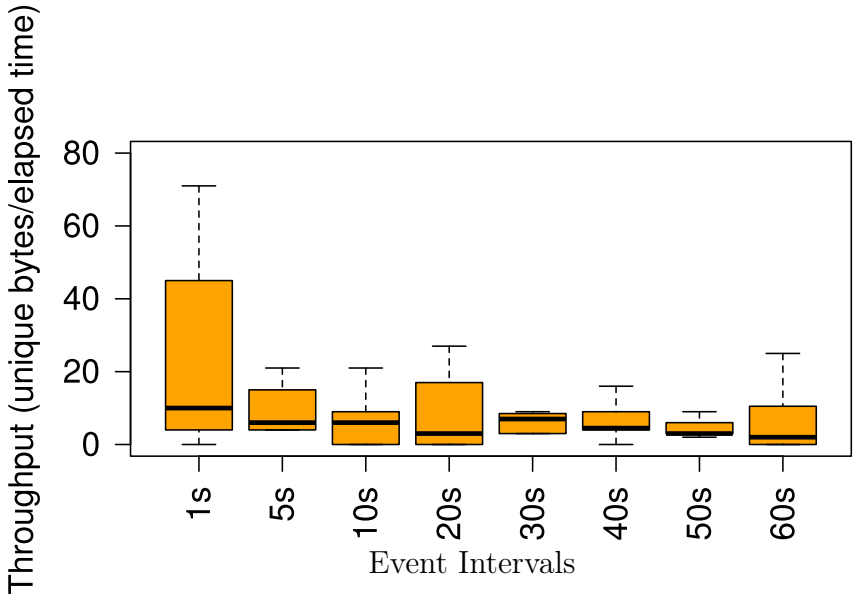



Figure 37: Throughput for client2 and for the Socketio based solution

value at 60s interval. This trend is because of the number of events generated at a particular interval.

- For Sharejs and Socket.io based solutions there are fluctuations in this metric value at all event intervals.
- The throughput is less in Socketio based solution compared to other solutions.

### 4.4.8 Qualitative Evaluation from programmer's perspective

The mutation-summary [48] library is useful to design screen sharing web applications. It can clone the entire DOM changes with minimal latency. Also, with the adequate documentation available, it is not difficult to design web applications using this library. However it has some limitations apart from the ones mentioned in [48], such as it cannot clone all the contents of certain DOM (Document Object Model) elements. As an example iframe content cannot be cloned. We found this when we tried to clone the embedded youtube iframe media content in our application, across the clients. It gives only read access to the content for the viewers and because of this limitation it cannot be used for designing interactive applications. To support user interactions, the library has to be modified as per the needs. It becomes complicated to modify the library if there are multiple complex CSS selectors such as pseudo element (for example "div:first-child"). The mutation-summary API [40] consists of a single class called "MutationSummary" that can be instantiated with the available configuration options to observe the changes in the Document Object Model (DOM). The observed net DOM changes are then passed to the callback function. This makes it easy to observe all the DOM changes in a single call. Also, the callback function is provided with only the net-effect of the DOM changes (for example when a new element is created and then deleted before the callback, the net effect is that nothing has changed and hence no changes to be sent). There are options to observe only elements, attributes, character data or all of them by setting the type of query in the class configuration. There are also methods such as 'reconnect' and 'disconnect' to dynamically observe or to stop observing changes respectively.
The ShareJS library [23] is useful to design real-time collaborative web applications such as text editors, multi-player games. There are some considerations for the programmers that we made during the implementation of the solution using this library. Our analysis is based on the library version 0.7 that we used for this thesis. The implementation considerations in this version of the library are:

- Due to the complexity involved in designing Operational Transform algorithms it is time consuming to implement them for the content other than text and JSON.
- There are some discrepancies in the information provided in the documentation due to which the programmer may need to spend time in fixing the issues.
- The older versions of the library are not suitable to design distributed web applications. That is to design web applications that can run in a distributed system where multiple components in a connected network communicate with each other. These web applications are used for different purposes such as load sharing, database servers, application servers and so on.

Apart from the considerations it is easy to implement the real-time web solutions using this library. Also, the latest version available is suitable for web applications suitable for distributed architecture.

The ShareJS server API gives flexibility to choose 'Redis' and 'livedb' in memory

datastore for its back-end and the default one being 'livedb'. For client-server communication it supports browserchannel [29], websockets or any other transport protocol that not only guarantees in order message delivery but also provides websocket-like API to client and node object stream to the server. However, the recommended one is browserchannel as some of the features can be used without writing an extra bit of code to call additional methods (i.e., to re-establish session state of the socket on reconnect 'socket.onopen' method is to be called, to allow data transfer during connection establishment 'socket.canSendWhileConnecting = true' flag is to be set, and for JSON stringifying 'socket.canSendJSON = true' is to be set). Also, as per the documentation, Socketio is not recommended for the transport especially for collaborative web applications but it doesn't mention if this is true for the Socketio version 1.0.

The ShareJS client API provides methods to; open a connection to the sharejs server; create or edit documents for storing the client data; subscribe or unsubscribe document changes; submit or delete operations. However, these methods are frequently changed or removed entirely over the versions and the documentation for it is sometimes misleading. It is recommended to follow the changelog [4] for the changes in the latest versions. Also, as per the changelog version 0.7 is not a stable version yet and improvements are expected.

The Socket.io library [56] version 1.0 is useful to design real-time collaborative web applications. It is easy to implement real-time web applications using this library. The documentation is adequate to understand the library functions. One requirement in this library is that both the server and the client must be using the socket.io libraries for establishing connection. Both the client and server APIs are easy to use that comes with proper documentation. The server API provides flexibility to use any in memory based datastore. The currently available options are redis, mongodb, and default socket.io adapter are supported. The default in memory based adapter is called 'socket.io-adapter' and the redis adapter is called 'socket.io-redis' that is built on top of default socket.io adapter. The client API provides an option for the client to reconnect automatically based on the type of reconnection i.e., reconnect after some delay, maximum reconnection delay and so on. The socket.io library allows non-socket processes to communication with socket.io servers using 'socket.io-emitter' library. It is recommended to check on all the improvements from the blog page of the Socket.io website [57]. The limitation in socket.io is that it does not establish a connection when self-signed certificates are used.

# 5 Conclusion

This thesis realized an efficient web-based solution for remote management of pervasive displays. We leveraged the cloud computing paradigm and deployed a management server that clones the state changes in the content to the pervasive displays, instead of sending the entire content for every state change. We adopted HTML5 and WebSocket to provide low-latency bidirectional communication. We implemented three versions of a representative digital signage application using state-of-the-art JavaScript libraries for real-time communication and evaluated the performance of each of them. Moreover, we also analyzed the performance of the VNC with our solution for comparison purposes.

We conducted several experiments to study the performance of the application for each of the considered libraries. We focused on performance metrics such as payload, round trip time (RTT), application response time, and page load time. We studied these metrics in different network conditions. From the experiments conducted, we conclude that the solution based on mutation-summary has less application response time but slightly higher payload size and RTT values compared to other solutions. Moreover, the solution based on mutation-summary has overall a higher page load time. From these observations we suggest that the mutation-summary-based solution is suited for non-interactive applications. The solutions based on ShareJS and Socket.io are better suited for real-time collaborative applications. However, the delay is higher thus making them not the best choice for digital signage scenarios. There is a trade-off between choosing libraries for real-time applications requiring user interactions and the applications requiring less delay with no user interactions.

We also experimented with other libraries such as React.js and DerbyJS. One observation made out of these libraries is that we can build an effective real-time collaborative web application using ShareJS to observe the state changes of the Document Object Model (DOM) content and React.js to render the content on the target machine. We argue that this solution would be effective in terms of delay and overhead.

We plan to investigate further on effective web based solutions for remote management of pervasive displays. One aspect would be to provide solutions for the remote management of multiple content shown at the same time on pervasive displays. We wish to explore effective solutions to automate error detection and recovery of pervasive displays.

# References

[1] Adam Bergkvist, D Burnett, and Cullen Jennings. "A. Narayanan," WebRTC 1.0: Real-time Communication Between Browsers". In: *World Wide Web Consortium WD WD-webrtc-20120821* (2012).

[2] Robin Berjon. *W3C HTML5 Working Draft*. 2012.

[3] *Boston to a T: July 2011*. URL: `http://bostontoat.blogspot.fi/2011_07_01_archive.html` (visited on 10/29/2015).

[4] *Changelog:share/ShareJS Wiki*. URL: `https://github.com/share/ShareJS/wiki/Changelog` (visited on 10/30/2015).

[5] Byung-Gon Chun, Sunghwan Ihm, Petros Maniatis, Mayur Naik, and Ashwin Patti. "Clonecloud: elastic execution between mobile device and cloud". In: *Proceedings of the sixth conference on Computer systems*. ACM. 2011, pp. 301–314.

[6] Sarah Clinch, Mateusz Mikusz, Miriam Greis, Nigel Davies, and Adrian Friday. "Mercury: an application store for open display networks". In: *Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing*. ACM. 2014, pp. 511–522.

[7] Nigel Davies, Sarah Clinch, and Florian Alt. "Pervasive Displays: Understanding the Future of Digital Signage". In: *Synthesis Lectures on Mobile and Pervasive Computing* 8.1 (2014), pp. 1–128. DOI: `10.2200/S00558ED1V01Y201312MPC011`. eprint: `http://dx.doi.org/10.2200/S00558ED1V01Y201312MPC011`. URL: `http://dx.doi.org/10.2200/S00558ED1V01Y201312MPC011`.

[8] Nigel Davies, Marc Langheinrich, Rui Jose, and Albrecht Schmidt. "Open display networks: A communications medium for the 21st century". In: *Computer* 5 (2012), pp. 58–64.

[9] *Decision Mapper*. URL: `http://decisionmapper.com/tutorials/derby1` (visited on 09/14/2015).

[10] *derbyjs/racer · GitHub*. URL: `https://github.com/derbyjs/racer` (visited on 09/14/2015).

[11] Rachna Dhamija, J. D. Tygar, and Marti Hearst. "Why Phishing Works". In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '06. Montréal, Québec, Canada: ACM, 2006, pp. 581–590. ISBN: 1-59593-372-7. DOI: `10.1145/1124772.1124861`. URL: `http://doi.acm.org/10.1145/1124772.1124861`.

[12] *DOM Standard*. URL: `https://dom.spec.whatwg.org/#mutation-observers` (visited on 09/12/2015).

[13] Ivan Elhart, Marc Langheinrich, Nemanja Memarovic, and Tommi Heikkinen. "Scheduling Interactive and Concurrently Running Applications in Pervasive Display Networks". In: *Proceedings of The International Symposium on Pervasive Displays*. ACM. 2014, p. 104.

[14] C. A. Ellis and S. J. Gibbs. "Concurrency Control in Groupware Systems". In: *SIGMOD Rec.* 18.2 (June 1989), pp. 399–407. ISSN: 0163-5808. DOI: 10.1145/66926.66963. URL: http://doi.acm.org/10.1145/66926.66963.

[15] *enmasseio/timesync · GitHub*. URL: https://github.com/enmasseio/timesync (visited on 09/14/2015).

[16] Aiman Erbad, Michael Blackstock, Adrian Friday, Rodger Lea, and Jalal Al-Muhtadi. "Magic broker: A middleware toolkit for interactive public displays". In: *Pervasive Computing and Communications, 2008. PerCom 2008. Sixth Annual IEEE International Conference on*. IEEE. 2008, pp. 509–514.

[17] *FAQ - Frequently Asked Questions | MQTT*. URL: http://mqtt.org/faq (visited on 10/30/2015).

[18] *Faye: Simple pub/sub messaging for the web*. URL: http://faye.jcoglan.com/ (visited on 09/22/2015).

[19] *faye/faye-websocket-node · GitHub*. URL: https://github.com/faye/faye-websocket-node (visited on 09/22/2015).

[20] I Fette and A Melnikov. "Rfc 6455: The websocket protocol". In: *IETF, December* (2011).

[21] Joe Finney, Stephen Wade, Nigel Davies, and Adrian Friday. *FLUMP The FLexible Ubiquitous Monitor Project*. 1996.

[22] B Fitzpatrick, B Slatkin, M Atkins, and J Genestoux. *PubSubHubbub Core 0.4. Working draft, PubSubHubbub W3C Community Group (2013)*.

[23] Joseph Gentle. *share/ShareJS · GitHub*. URL: https://github.com/share/ShareJS (visited on 09/12/2015).

[24] Tommi Heikkinen, Tomas Lindén, Timo Ojala, Hannu Kukka, Marko Jurmu, and Simo Hosio. "Lessons learned from the deployment and maintenance of UBI-hotspots". In: *Multimedia and Ubiquitous Engineering (MUE), 2010 4th International Conference on*. IEEE. 2010, pp. 1–6.

[25] Tommi Heikkinen, Petri Luojus, and Timo Ojala. "UbiBroker: event-based communication architecture for pervasive display networks". In: *Proceedings of the IEEE International Conference on Pervasive Computing and Communication Workshops (PERCOM Workshops' 14)*. 2014, pp. 512–518.

[26] *Infrared touch manufacturer*. URL: http://www.huitoo.cn/huitoo/en/Product7000.asp (visited on 10/29/2015).

[27] Byungil Jeong, Luc Renambot, Ratko Jagodic, Rajvikram Singh, Julieta Aguilera, Andrew Johnson, and Jason Leigh. "High-performance dynamic graphics streaming for scalable adaptive graphics environment". In: *SC 2006 Conference, Proceedings of the ACM/IEEE*. IEEE. 2006, pp. 24–24.

[28] Brad Johanson, Shankar Ponnekanti, Caesar Sengupta, and Armando Fox. "Multibrowsing: Moving web content across multiple displays". In: *Ubicomp 2001: Ubiquitous Computing*. Springer. 2001, pp. 346–353.

[29] *josephg/node-browserchannel · GitHub*. URL: https://github.com/josephg/node-browserchannel (visited on 09/14/2015).

[30] *json0/README.md at master · ottypes/json0 · GitHub*. URL: https://github.com/ottypes/json0/blob/master/README.md (visited on 09/13/2015).

[31] Kai Kuikkaniemi, Vilma Lehtinen, Matti Nelimarkka, Max Vilkki, Jouni Ojala, and Giulio Jacucci. "Designing for presenters at public walk-up-and-use displays". In: *Proceedings of the 8th International Conference on Tangible, Embedded and Embodied Interaction*. ACM. 2014, pp. 225–232.

[32] Karthik Kumar and Yung-Hsiang Lu. "Cloud computing for mobile users: Can offloading computation save energy?" In: *Computer* 4 (2010), pp. 51–56.

[33] Tomas Linden, Tommi Heikkinen, Timo Ojala, Hannu Kukka, and Marko Jurmu. "Web-based Framework for Spatiotemporal Screen Real Estate Management of Interactive Public Displays". In: *Proceedings of the 19th International Conference on World Wide Web*. WWW '10. Raleigh, North Carolina, USA: ACM, 2010, pp. 1277–1280. ISBN: 978-1-60558-799-8. DOI: 10.1145/1772690.1772901. URL: http://doi.acm.org/10.1145/1772690.1772901.

[34] *loadfocus/pageloadtime · GitHub*. URL: https://github.com/loadfocus/pageloadtime (visited on 09/14/2015).

[35] *macbre/phantomas*. URL: https://github.com/macbre/phantomas (visited on 10/27/2015).

[36] *macbre/phantomas · GitHub*. URL: https://github.com/macbre/phantomas (visited on 09/14/2015).

[37] J Martin. *noVNC project website*. 2011.

[38] D Mills, J Martin, J Burbank, and W Kasch. "RFC 5905: Network Time Protocol version 4: Protocol and algorithms specification". In: *Internet Engineering Task Force* (2010).

[39] Gabriel L. Muller. "HTML5 WebSocket protocol and its application to distributed computing". In: *CoRR* abs/1409.3367 (2014). URL: http://arxiv.org/abs/1409.3367.

[40] *mutation-summary/APIReference.md at master · rafaelw/mutation-summary*. URL: https://github.com/rafaelw/mutation-summary/blob/master/APIReference.md (visited on 10/29/2015).

[41] Matei Negulescu and Yang Li. "Open Project: a lightweight framework for remote sharing of mobile applications". In: *Proceedings of the 26th annual ACM symposium on User interface software and technology*. ACM. 2013, pp. 281–290.

[42] Elena Oat, Mario Di Francesco, and Tuomas Aura. "MoCHA: Augmenting Pervasive Displays through Mobile Devices and Web-based Technologies". In: *The 1st IEEE Workshop on Developing Applications for Pervasive Display Networks (PD-Apps '14)*. Mar. 2014, pp. 506–511. URL: http://uc.inf.usi.ch/sites/all/files/pd-apps2014/papers/1569868391.pdf.

[43] Simon Olberding, Jürgen Steimle, Suranga Nanayakkara, and Pattie Maes. "CloudDrops: Stamp-sized Pervasive Displays for Situated Awareness of Web-based Information". In: ().

[44] *Operational transformation - Wikipedia, the free encyclopedia*. URL: https://en.wikipedia.org/wiki/Operational_transformation (visited on 09/13/2015).

[45] Addy Osmani. *Detect, Undo And Redo DOM Changes With Mutation Observers*. URL: http://addyosmani.com/blog/mutation-observers/ (visited on 09/12/2015).

[46] Shawn Ostermann. *Tcptrace*. 2005.

[47] *PhantomJS | PhantomJS*. URL: http://phantomjs.org/ (visited on 09/14/2015).

[48] *rafaelw/mutation-summary*. URL: https://github.com/rafaelw/mutation-summary (visited on 09/12/2015).

[49] *Reconciliation | React*. URL: https://facebook.github.io/react/docs/reconciliation.html (visited on 09/14/2015).

[50] *Redis*. URL: http://redis.io/ (visited on 09/13/2015).

[51] Alex Russell, Greg Wilkins, David Davis, and Mark Nesbitt. *The bayeux specification*. 2013.

[52] *Screensharing a browser tab in HTML5? - HTML5 Rocks*. URL: http://www.html5rocks.com/en/tutorials/streaming/screenshare/ (visited on 09/14/2015).

[53] Mohit Sethi, Elena Oat, Mario Di Francesco, and Tuomas Aura. "Secure Bootstrapping of Cloud-Managed Ubiquitous Displays". In: *The 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing (UbiComp 2014)*. Sept. 2014, pp. 739–750. DOI: 10.1145/2632048.2632049. URL: http://www.uta.edu/faculty/mariodf/documents/ubicomp-2014.pdf.

[54] *share/livedb · GitHub*. URL: https://github.com/share/livedb (visited on 09/14/2015).

[55] Pieter Simoens, Filip De Turck, Bart Dhoedt, and Piet Demeester. "Remote display solutions for mobile cloud computing". In: *Computer* 8 (2011), pp. 46–53.

[56] *Socket.IO*. URL: http://socket.io/ (visited on 09/14/2015).

[57] *Socket.IO — Introducing Socket.IO 1.0*. URL: http://socket.io/blog/introducing-socket-io-1-0/#future-innovation (visited on 10/30/2015).

[58] *Socket.IO — Rooms and Namespaces*. URL: http://socket.io/docs/rooms-and-namespaces/#sending-messages-from-the-outside-world (visited on 09/14/2015).

[59] *Socket.IO — Using multiple nodes*. URL: http://socket.io/docs/using-multiple-nodes/ (visited on 09/14/2015).

[60] *socketio/engine.io · GitHub*. URL: `https://github.com/socketio/engine.io#goals` (visited on 09/14/2015).

[61] Ariel Stolerman. "RFC 6143: The Remote Framebuffer (RFB) Protocol Analysis". In: (2013).

[62] Martin Strohbach and Miquel Martin. "Toward a platform for pervasive display applications in retail environments". In: *IEEE Pervasive Computing* 2 (2011), pp. 19–27.

[63] Sasu Tarkoma. *Publish/subscribe systems: design and principles*. John Wiley & Sons, 2012.

[64] *text/README.md at master · ottypes/text · GitHub*. URL: `https://github.com/ottypes/text/blob/master/README.md` (visited on 09/13/2015).

[65] *The future of web apps is – ready? – isomorphic JavaScript | VentureBeat | Dev | by J. O'Dell*. URL: `http://venturebeat.com/2013/11/08/the-future-of-web-apps-is-ready-isomorphic-javascript/` (visited on 09/14/2015).

[66] Mark Ethan Trostler. *Testable JavaScript*. " O'Reilly Media, Inc.", 2013.

[67] Cheng-Lin Tsao, Sandeep Kakumanu, and Raghupathy Sivakumar. "SmartVNC: an effective remote computing solution for smartphones". In: *Proceedings of the 17th annual international conference on Mobile computing and networking*. ACM. 2011, pp. 13–24.

[68] *WAMP - Web Application Messaging Protocol*. URL: `http://wamp-proto.org/` (visited on 10/30/2015).

[69] David Wang, Alex Mah, and S Lassen. "Google wave operational transformation". In: *Whitepaper, Google Inc* (2010).

[70] Vanessa Wang, Frank Salim, and Peter Moskovits. *The definitive guide to HTML5 WebSocket*. Vol. 1. Springer, 2013.

[71] *WebRTC Tab Content Capture - The Chromium Projects*. URL: `http://www.chromium.org/developers/design-documents/extensions/proposed-changes/apis-under-development/webrtc-tab-content-capture` (visited on 09/14/2015).

[72] *wesleyhales/loadreport · GitHub*. URL: `https://github.com/wesleyhales/loadreport` (visited on 09/14/2015).

[73] Lauren Wood, Vidur Apparao, Laurence Cable, Mike Champion, Mark Davis, Joe Kesselman, Tom Pixley, Jonathan Robie, Peter Sharpe, and Chris Wilson. "Document object model (dom) level 2 specification". In: *World Wide Web Consortium. www. w3. org/TR/DOM-Level-2* (2000).

**APPENDIX**

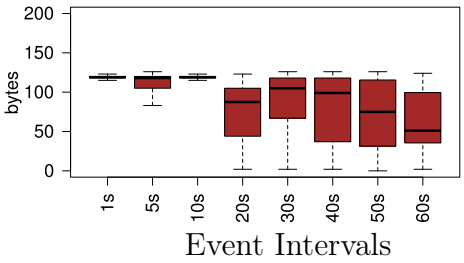
# A Ethernet Experiment Related Plots

We also considered Ethernet interface for our experiment and the relevant details are mentioned in this section.

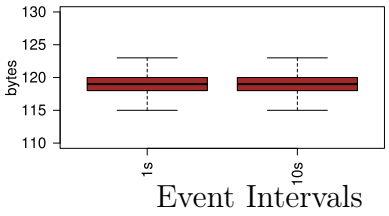
| Interface | Download speed | Upload speed |
|---|---|---|
| Ethernet | 94.89 Megabits per second | 93.56 Megabits per second |

Table 2: Ethernet Interface Specifications

The plot of WebSocket Frames payload metric for the mutation-summary based solution is shown in Figure 38a. From the box-plot we observe that there is a variation in the median value over the event intervals. The Figure 38b is a clear representation of the box-plot for 1s and 5s event intervals of the main plot. The distribution of data is skewed left at all the event intervals except at 60s where it has slightly skewed right distribution. At 1s, 10s and 50s event intervals data is symmetrically distributed and at 60s data is skewed right. The inter-quartile range value is approximately between 40-80 bytes over the intervals 20s, 30s, 40s, 50s, and 60s and has a value between 3-10 bytes over the intervals 1s, 5s, and 10s.




(a) For all the intervals




(b) For the intervals 1s and 10s

Figure 38: WebSocket Frames Payload for the solution based on mutation-summary library

The plot of WebSocket Frames payload metric for the ShareJS based solution is shown in Figure 39. From the box-plot we observe that the payload median value is about same i.e., 20 bytes at all the intervals except at 60s that has a value of 40 bytes. The distribution of data is skewed right at all the intervals. The inter-quartile range value is approximately between 75-85 bytes over the intervals.

The plot of WebSocket Frames payload metric for the Socketio based solution is shown in Figures 40a and 40b. From the box-plot we observe that the payload median value has a large variation over the event intervals. The distribution of data is skewed left at 1s, 5s, 20s, 30s intervals and is skewed right at 40s, 50s, and 60s

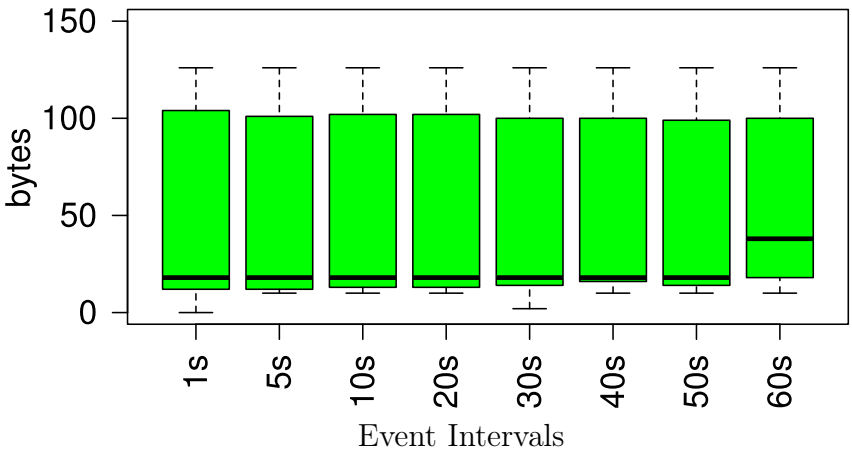
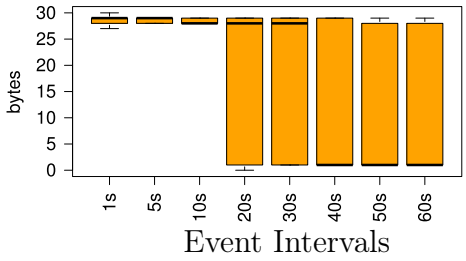
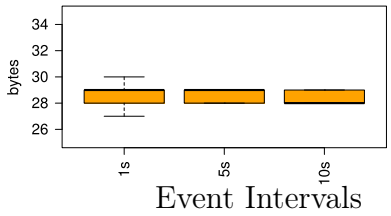



Figure 39: WebSocket Frames Payload for the solution based on ShareJS library

intervals. The inter-quartile range value is approximately between 1 byte at the intervals 1s, 5s, 10s and has a value of 25 bytes approximately at the other intervals.




(a) For all the intervals

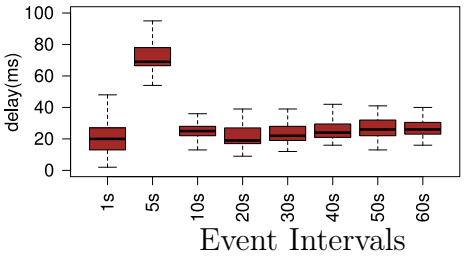



(b) For intervals 1s, 5s, 10s

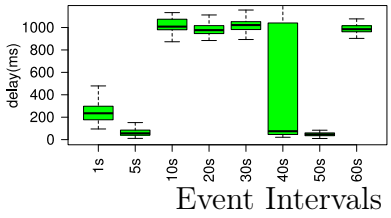
Figure 40: WebSocket Frames Payload for the solution based on Socketio library

The plot of application response time metric for the mutation-summary based solution is shown in Figure 41a. From the box-plot median value of the metric is between 20-25 ms approximately at all the event intervals except at 5s that has a value of 65 ms approximately. The distribution of data is skewed right at all the intervals except at 10s that has symmetric distribution of data. The inter-quartile range value is between 4-16 ms approximately at all the intervals.

The plot of application response time metric for the ShareJS based solution is shown in Figure 41b. The distribution of data is skewed right at all the intervals except at 30s that has symmetric distribution. The inter-quartile range value is between 40-120 ms approximately at all the intervals except at 40s that has a value of 1000 ms.



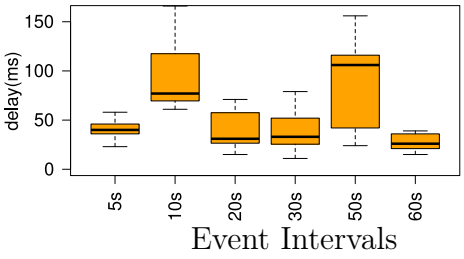

(a) Mutation-summary based solution



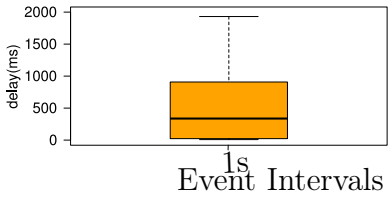

(b) ShareJS based solution

Figure 41: Application response time plots

The plot of application response time metric for the Socket.io based solution, is shown in Figures 42a and 42b. From the box-plot median value of the metric is between 30-110 ms at all the event intervals except at 1s that has a value of 300 ms. The distribution of data is skewed right at all the intervals except at 50s that have skewed left distribution. The inter-quartile range value is 900 ms at 1s interval and for the rest of the intervals it is between 13-70 ms approximately.




(a) For intervals 5s, 10s, 20s, 30s, 50s, 60s




(b) For 1s interval

Figure 42: Application response time for the solution based on Socketio library

The plot of Round-Trip Time (RTT) average value for client 1 and for all the solutions, is shown in Figure 43. From the box-plot the approximate median values of the metric data for the mutation-summary, ShareJS and Socket.io based solutions over the intervals are in the range 9.3-10 ms, 8.5-9 ms, and 8.4-9.4 ms respectively. The approximate inter-quartile values in mutation-summary, ShareJS and Socket.io based solutions over the intervals are in the range 1.2-2.3 ms, 1-2.3 ms, 0.2-2.2 ms respectively. The distribution of data for the solutions is skewed right at all intervals.

The plot of Round-Trip Time (RTT) average value for client 2 and for all the solutions, is shown in Figure 44. From the box-plot the approximate median values of the metric data for the mutation-summary, ShareJS and Socket.io based solutions over the intervals are in the range 9.8-10 ms, 8.9-10.4 ms, and 8.8-11 ms respectively. The approximate inter-quartile values in mutation-summary, ShareJS and Socket.io based solutions over the intervals are in the range 0.7-2.2 ms, 2.5-3.3 ms, 0.7-4.5 ms

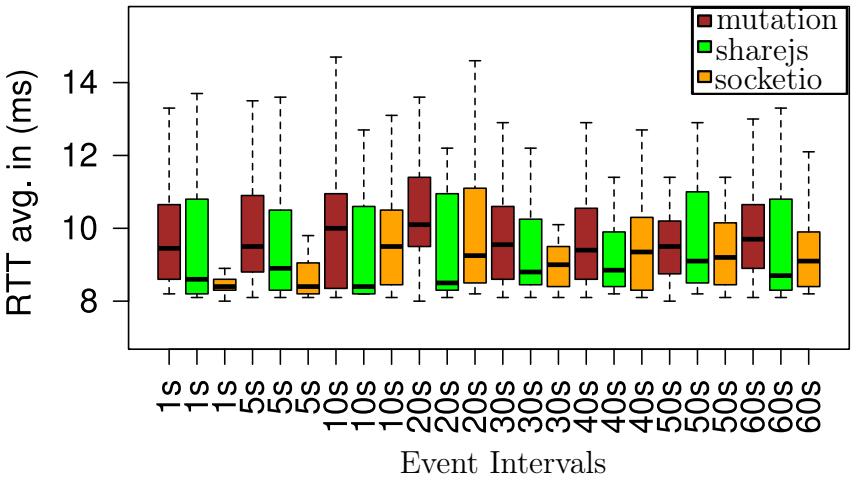



Figure 43: RTT average for client 1

respectively. The distribution of data for the solutions is skewed right at all intervals except at 60s that has a skewed left distribution for Socket.io solution.

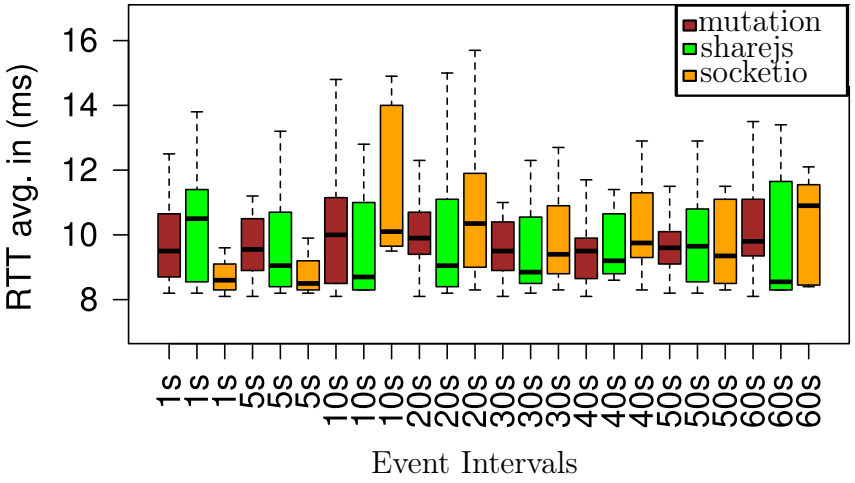



Figure 44: RTT average for client 2

The plot of Round-Trip Time (RTT) maximum value for client 1 and for all the solutions, is shown in Figure 45. From the box-plot the approximate median values of the metric data for the mutation-summary, ShareJS and Socket.io based solutions over the intervals are between 14-16 ms, around 10 ms, and between 8-14 ms respectively. The approximate inter-quartile values in mutation-summary, ShareJS and Socket.io based solutions over the intervals are in the range 4-10 ms, 5-20 ms,

2-37 ms respectively. The distribution of data for the solutions is skewed right at all the intervals.

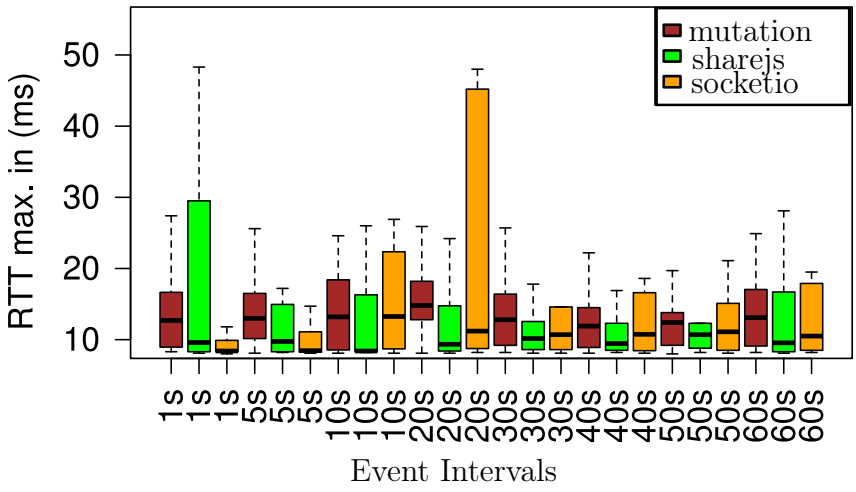



Figure 45: RTT maximum for client 1

The plot of Round-Trip Time (RTT) maximum value for client 2 and for all the solutions, is shown in Figure 46. From the box-plot the approximate median values of the metric data for the mutation-summary, ShareJS and Socket.io based solutions over the intervals are in the range 12-15 ms, 9-18 ms, and 10-30 ms respectively. The approximate inter-quartile values in mutation-summary, ShareJS and Socket.io based solutions over the intervals are in the range 3-7 ms, 3-15 ms, 3-38 ms respectively. The distribution of data for the solutions is skewed right at all the intervals except at 5s/10s/50s. At 5s and 50s interval mutation summary based solution has symmetric distribution and at the 10s interval Socket.io based solution has symmetric distribution.

The plot of Round-Trip Time (RTT) minimum value for client 1 and for all the solutions, is shown in Figure 47. From the box-plot the approximate median values of the metric data for the mutation-summary, ShareJS and Socket.io based solutions over the intervals are in the range 8.1-8.2 ms, 8.1-8.3 ms, and 8.1-8.2 ms respectively. The approximate inter-quartile values in mutation-summary, ShareJS and Socket.io based solutions over the intervals are in the range 0.2-0.3 ms, 0.1-0.2 ms, 0.1-0.3 ms respectively. The distribution of data for the solutions is symmetric except at 60s interval that has skewed right distribution.

The plot of Round-Trip Time (RTT) minimum value for client 2 and for all the solutions, is shown in Figure 48. From the box-plot the approximate median values of the metric data for the mutation-summary, ShareJS and Socket.io based solutions over the intervals are in the range 8.05-8.2 ms, 8.2-8.4 ms, and 8.15-8.3 ms respectively. The approximate inter-quartile values in mutation-summary, ShareJS and Socket.io based solutions over the intervals are in the range 0.2-0.3 ms, 0.1-0.3

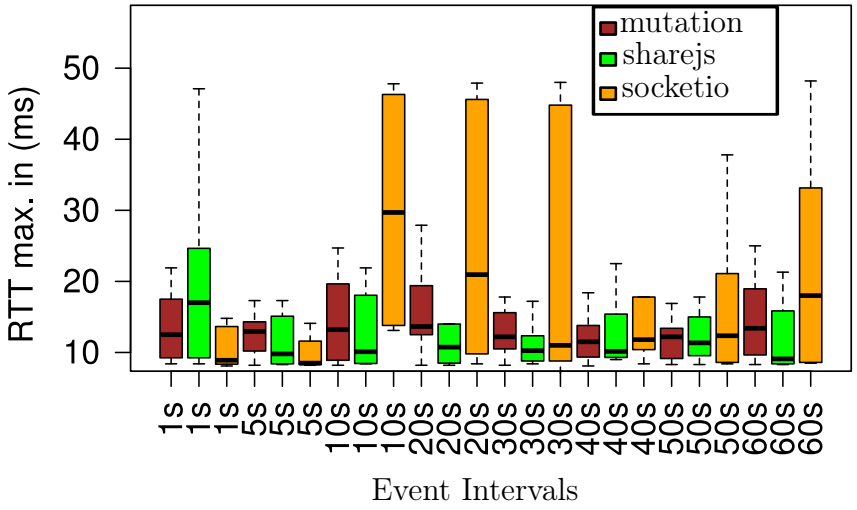



Figure 46: RTT maximum for client 2

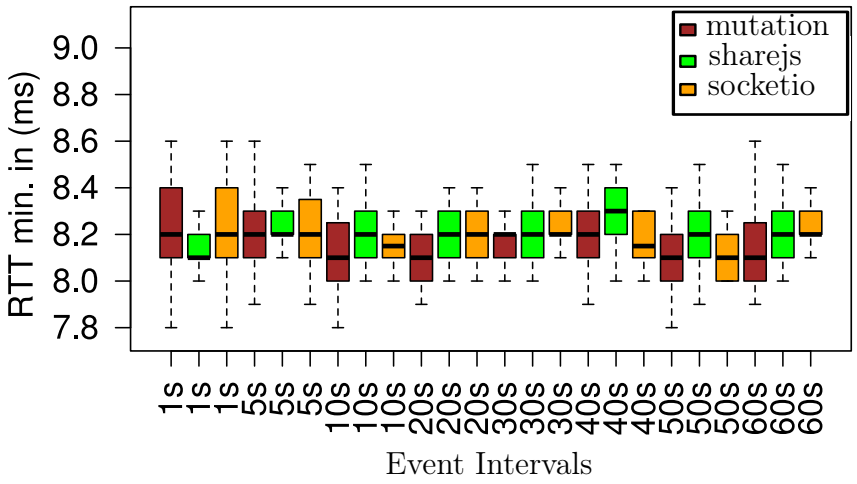



Figure 47: RTT minimum for client 1

ms, 0.1-0.3 ms respectively. The distribution of data for the solutions is symmetric for most of the intervals.

The plot of unique data bytes value for the mutation-summary based solution, is shown in Figure 49a. From the box-plot median value of the metric is about 600 bytes at all the event intervals. The distribution of data is skewed right at all the event intervals. The inter-quartile range value is between 30-400 bytes approximately at all the intervals.

The plots of unique data bytes value for the ShareJS based solution, is shown in Figures 50a and 50b. From the box-plot median value of the metric is between

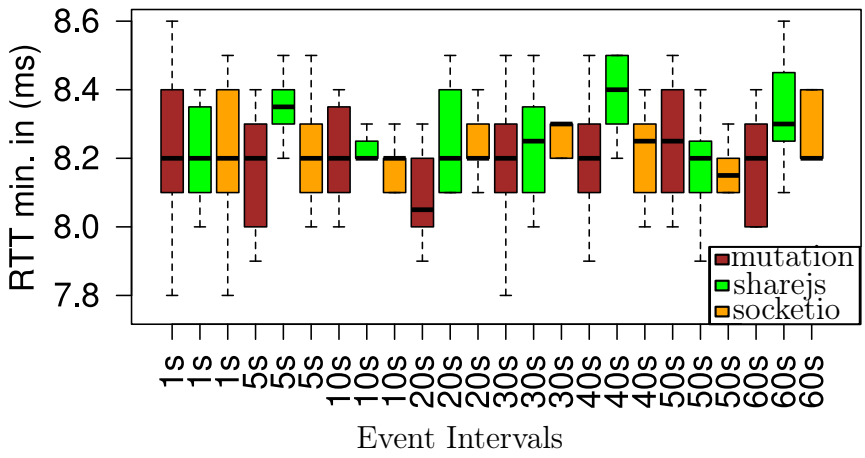
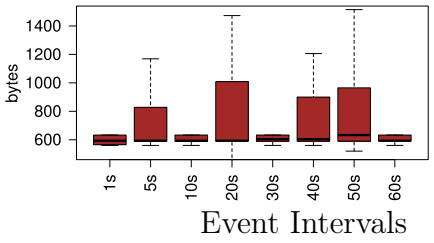
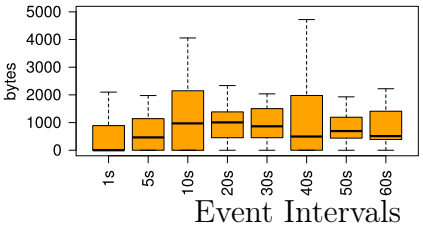



Figure 48: RTT minimum for client 2




(a) Mutation-summary based solution




(b) Socketio based solution

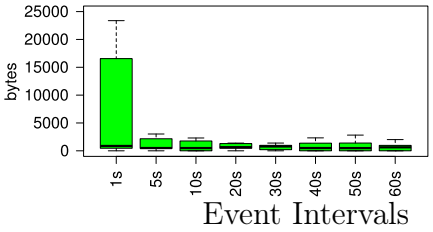
Figure 49: Unique Bytes in TCP payload

500-1000 bytes at all the event intervals. The inter-quartile range value is between 500-15000 bytes approximately at all the intervals.

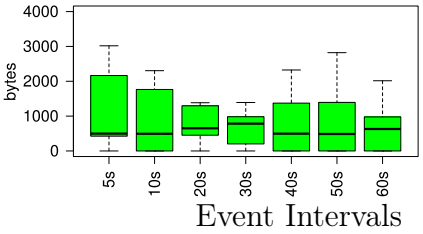
The plot of unique data bytes value for the Socket.io based solution, is shown in Figure 49b. From the box-plot median value of the metric is between 0-1000 bytes at all the event intervals. The distribution of data is skewed right at all the event intervals. The inter-quartile range value is between 800-2000 bytes approximately at all the intervals.

The plot of actual data packets value for the mutation-summary based solution, is shown in Figure 51a. From the box-plot median value of the metric is consistent, that is about 2 packets at all the event intervals. The distribution of data is skewed right at all the event intervals. The inter-quartile range value is between 2-3 packets at all the intervals.

The plot of actual data packets value for the ShareJS based solution, is shown in Figure 51b. From the box-plot median value of the metric is about 2 packets at all the event intervals. The distribution of data is skewed right at all the event



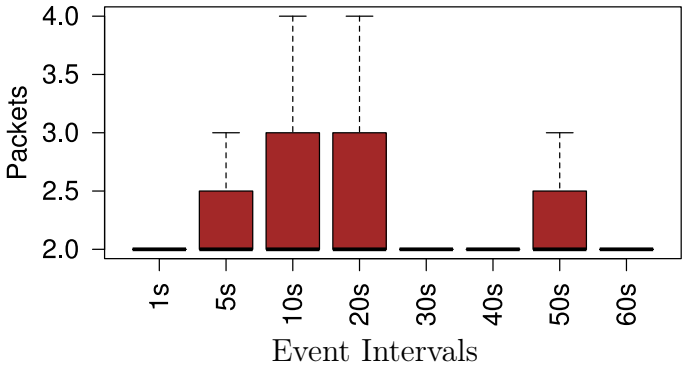

(a) For intervals 1s, 5s, 10s, 20s, 40s, 50s, 60s respectively



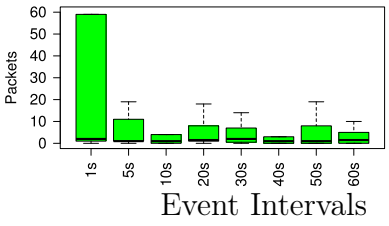

(b) For interval 30s respectively

Figure 50: Unique Bytes in TCP payload for the ShareJS based solution

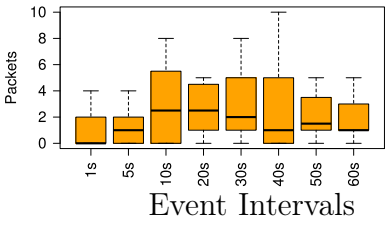
intervals. The inter-quartile range value is between 2-10 packets approximately at all the intervals except at 1s interval where the value is about 59 packets.




(a) Actual Packets in TCP payload for the mutation-summary based solution




(b) ShareJS based solution



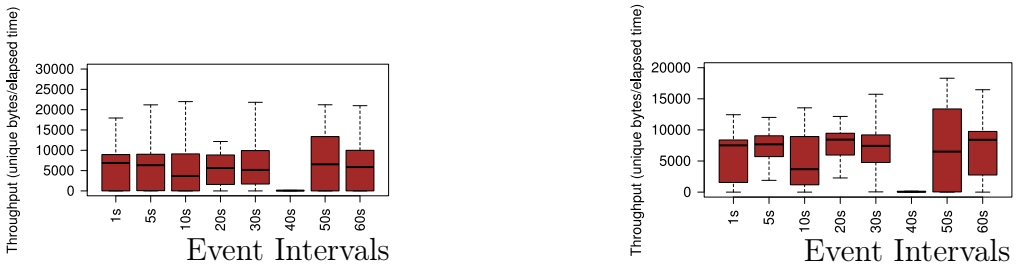

(c) Socketio based solution

Figure 51: Actual Packets in TCP payload

The plot of actual data packets value for the Socket.io based solution, is shown in Figure 51c. From the box-plot median value of the metric is between 0-3 packets at all the event intervals. The distribution of data is skewed right at all the event intervals except at 20s where it is symmetric. The inter-quartile range value is

between 2-5 packets approximately at all the intervals.

The plot of throughput value for client 1 and for the mutation-summary based solution, is shown in Figure 52a. From the box-plot median value of the metric is between 0-7000 bytes/second at all the event intervals. The distribution of data is skewed right at all the event intervals except at 20s and 40s intervals where the distribution is symmetric. The inter-quartile range value is between 0-14000 bytes/second approximately at all the intervals.

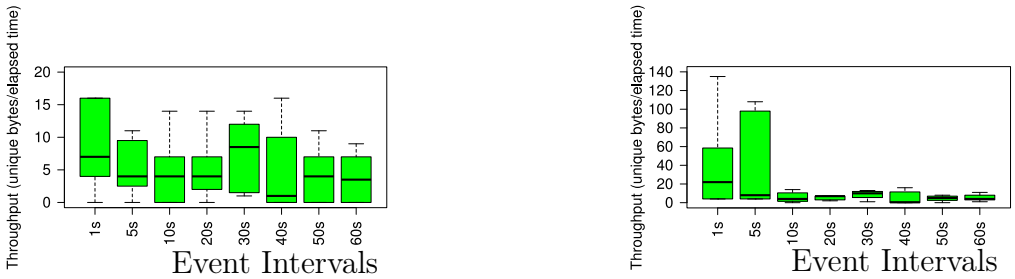



(a) For client1 (b) For client2

Figure 52: Throughput for the mutation-summary based solution

The plot of throughput value for client 2 and for the mutation-summary based solution, is shown in Figure 52b. From the box-plot, median value of the metric is between 0-9000 bytes/second at all the event intervals. The distribution of data is skewed right at all the event intervals except at 60s where the distribution is skewed left. The inter-quartile range value is between 3000-14000 bytes/second approximately at all the intervals.

The plot of throughput value for client 1 and for the ShareJS based solution, is shown in Figure 53a. From the box-plot, median value of the metric is between 1-9 bytes/second at all the event intervals. The distribution of data is skewed right at all the event intervals except at 30s where the distribution is skewed left. The inter-quartile range value is between 5-12 bytes/second approximately at all the intervals.



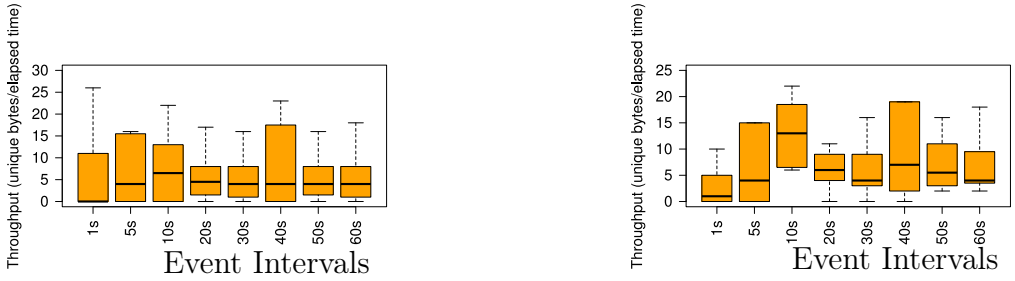

(a) For client1 (b) For client2

Figure 53: Throughput for the ShareJS based solution

The plot of throughput value for client 2 and for the ShareJS based solution, is shown in Figure 53b. From the box-plot, median value of the metric is between 0-20 bytes/second at all the event intervals. The distribution of data is skewed right at

all the event intervals except at 20s and 30s interval where the distribution is skewed left. The inter-quartile range value is between 7-95 bytes/second approximately at all the intervals.

The plot of throughput value for client 1 and for the Socket.io based solution, is shown in Figure 54a. From the box-plot, median value of the metric is between 0-7 bytes/second at all the event intervals. The distribution of data is skewed right at all the event intervals. The inter-quartile range value is between 6-17 bytes/second approximately at all the intervals.




(a) For client1

(b) For client2

Figure 54: Throughput for the ShareJS based solution

The plot of throughput value for client 2 and for the Socket.io based solution, is shown in Figure 54b. From the box-plot, median value of the metric is between 1-13 bytes/second at all the event intervals. The distribution of data is skewed right at all the event intervals except at 10s and 20s intervals where the distribution is symmetric. The inter-quartile range value is between 5-16 bytes/second approximately at all the intervals.